COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Benoit COQUERET

On the Complementarity of Software and Side-Channel Attacks Against Deep Learning Algorithms

Thèse présentée et soutenue à Rennes, le 17 décembre 2025
Unité de recherche : Centre Inria de l'Université de Rennes et Thales

Rapporteurs avant soutenance :

Alberto Bosio Professeur, Ecole Centrale Lyon, INL
Cédric Killian Professeur, Université de Saint-Étienne, LabHC

Composition du Jury :

Président·e :		*(à préciser après la soutenance)*
Examinateur·trice·s :	Alberto Bosio	Professeur, Ecole Centrale Lyon, INL
	Cédric Killian	Professeur, Université de Saint-Étienne, LabHC
	Christina Boura	Professeure, Université Paris Cité, IRIF
	Teddy Furon	Directeur de Recherche, centre Inria de l'Université de Rennes
	Pierre-Alain Moellic	Ingénieur de recherche, CEA Leti, Gardanne
Dir. de thèse :	Olivier Sentieys	Directeur de Recherche, centre Inria de l'Université de Rennes
Co-dir. de thèse :	Gabriel Zaid	Ingénieur de recherche, Thales, Toulouse
	Mathieu Carbone	Ingénieur de recherche, Thales, Toulouse

Résumé en Français

Au cours de la dernière décennie, les algorithmes d'*apprentissage profond* ont commencé à devenir de plus en plus communs dans de nombreux domaines. Comme ces algorithmes ont pour objectif de simuler l'intelligence, ceux-ci appartiennent au domaine de l'*intelligence artificielle* (IA). Plus précisément, comme ces algorithmes ont pour objectif d'apprendre automatiquement à résoudre une tâche en interagissant avec un ensemble de données, ils correspondent aux algorithmes d'*apprentissage automatique*. L'apprentissage profond représente donc un sous-domaine de l'apprentissage automatique, lui-même faisant partie de l'IA, comme le montre la Figure 1.

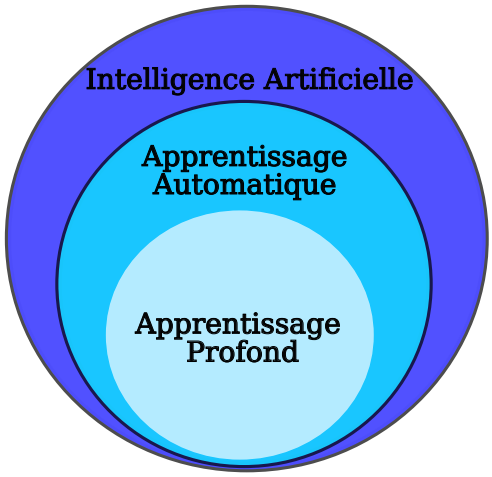


Figure 1 – Relation entre les domaines de l'apprentissage profond, de l'apprentissage automatique et de l'intelligence artificielle.

La spécificité de l'apprentissage profond réside dans son inspiration et son objectif d'imiter les fonctions cérébrales en créant des *réseaux de neurones artificiels* composés d'un grand nombre de *perceptrons* organisés en réseaux imitant les neurones et les synapses du cerveau. Si les premiers travaux sur ce sujet remontent à 1958, avec Rosenblatt *et al.* [Ros58] décrivant la structure du perceptron, l'apprentissage profond n'a gagné en popularité que récemment. En effet, les premiers algorithmes étaient limités dans leur capacité à apprendre des fonctions complexes, ce qui réduisait leur intérêt pour les tâches réalistes. Cependant, les améliorations apportées par l'algorithme de rétropropagation [RHW86]

et l'augmentation de la disponibilité des données combinées à l'augmentation de la puissance de calcul grâce à l'utilisation de *processeurs graphiques* ont permis des améliorations remarquables dans les performances de l'apprentissage profond. En particulier, en 2012, Alex Krizhevsky *et al.* [KSH12] ont réduit l'erreur des cinq meilleures prédictions sur le jeu de données ImageNet [Den+09] à 15.3%, soit 10.8% de mieux que le deuxième meilleur algorithme de l'époque, marquant ainsi le début de l'ère des réseaux de neurones. Depuis lors, ces algorithmes sont devenus la norme dans des domaines aussi variés que la conduite autonome, la détection de cancer ou la classification des logiciels malveillants. Cela se traduit clairement par une augmentation du nombre de publications mentionnant l'apprentissage profond dans leur résumé dans ces domaines, comme le montre la Figure 5.

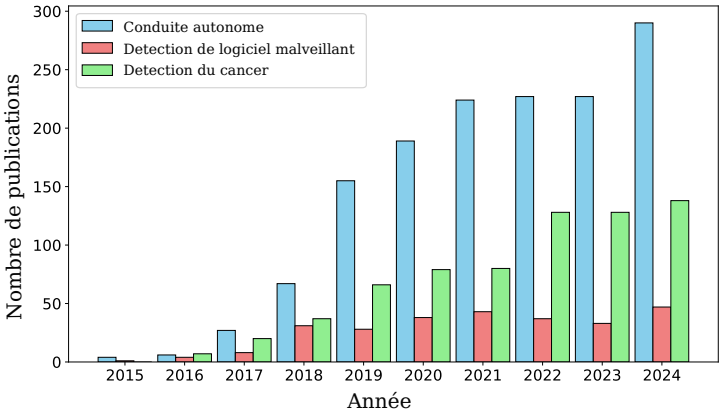


Figure 2 – Nombre de publications mentionnant l'apprentissage profond dans différents domaines sur ArXiv (Note: Le sujet, apprentissage profond et conduite autonome, par exemple, doit être explicitement mentionné dans les méta-données de l'article).

Cependant, avec ce succès, la *propriété intellectuelle* concernant les algorithmes d'apprentissage ainsi que celle de leurs données est devenue un sujet critique, suscitant diverses préoccupations concernant la sécurité de ces algorithmes.

En effet, la complexité accrue des tâches, combinée au temps et aux ressources nécessaires pour entraîner un *réseau de neurones profond*, a rendu les modèles entraînés très précieux. En conséquence, des attaques par extraction de modèle, c'est-à-dire des attaques visant à extraire des modèles afin d'acquérir une copie sans avoir à effectuer le processus d'entraînement, ont fait leur apparition, ainsi que des attaques visant à récupérer directement leurs données d'entraînement. Une autre menace à prendre en compte est qu'un attaquant pourrait choisir de cibler le processus décisionnel de l'IA. Dans les domaines critiques que nous avons mentionnés précédemment, cela pourrait avoir des conséquences

désastreuses. Au cours des dernières années, de nombreuses attaques de ce type contre les réseaux de neurones ont été proposées par des chercheurs, comme le montre la Figure 3.

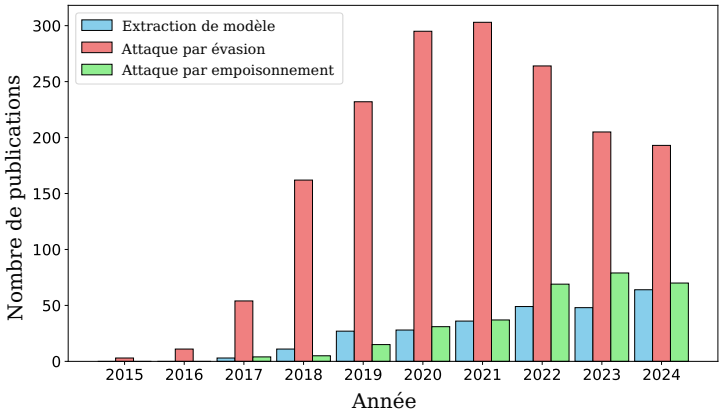


Figure 3 – Nombre de publications traitant d'une attaque contre les algorithmes basés sur l'apprentissage profond (Note: Le sujet, apprentissage profond et empoisonnement, par exemple, doit être explicitement mentionné dans les méta-données de l'article).

L'attaque la plus populaire et la plus étudiée contre les réseaux de neurones correspond aux attaques par évasion [GSS14; CW16], dans lesquelles un attaquant tente de créer une entrée qui sera mal classée par le DNN ciblé, tout en minimisant les modifications apportées à l'entrée d'origine. Bien que cette attaque se produise durant l'inférence, ce type de modèle est également vulnérable pendant son apprentissage. En effet, il est possible d'introduire un biais dans les données, c'est-à-dire de les corrompre, réduisant ainsi les performances du modèle et introduisant potentiellement des portes dérobées. Toutes ces attaques mettent en évidence la dualité des menaces qui pèsent sur les réseaux de neurones. D'une part, il faut tenir compte des attaques qui menacent la confidentialité du modèle et de ses données. Et d'autre part, il est également essentiel de considérer le modèle comme faisant partie d'un tout et de traiter les attaques qui tentent de perturber son processus décisionnel.

Enfin, à l'instar des composants cryptographiques, le modèle ne doit pas être considéré seulement comme un algorithme isolé, mais aussi comme faisant partie d'un ensemble implémenté et exécuté sur un système physique. Cette implémentation peut être défectueuse ou altérée, ce qui peut permettre d'autres types d'attaques, telles que les attaques physiques. Ce type d'attaque utilise des grandeurs physiques observables ou modifiables par un attaquant pour déduire certaines informations sur le traitement de l'algorithme de manière passive, ou pour altérer le comportement nominal des algorithmes.

Si ces attaques ont été étudiées sur les systèmes cryptographiques au cours des vingt dernières années [Koc96; KJJ99; BCO04], ce n'est que récemment qu'elles ont été appliquées aux réseaux de neurones [Wei+18; Bat+19]. Depuis, de nombreuses attaques menaçant divers aspects de la sécurité de l'IA, notamment la confidentialité du modèle [Hec+23; Jou+23b] ou l'intégrité des prédictions [RHF19], ont émergé.

Cependant, malgré toutes ces préoccupations concernant la sécurité de l'IA, les performances de ces algorithmes, associées à leur popularité croissante, ont conduit à leur adoption généralisée. Par conséquent, il est nécessaire que les gouvernements et les industries fournissent des lignes directrices pour évaluer la sécurité des produits basés sur l'IA. Une première étape a été franchie avec l'adoption du règlement sur l'IA, "AI Act", votée en 2024 par le Parlement européen [1]. La prochaine étape consiste à proposer un ensemble de règles pour certifier la sécurité des produits basés sur l'IA. En matière de cybersécurité, cet ensemble de règles de certification est défini par les schémas de certification et concerne tous les aspects de la cybersécurité, c'est-à-dire les attaques par canaux auxiliaires, la cryptographie, les attaques logicielles, etc. Le schéma le plus courant est celui des Critères communs pour l'évaluation de la sécurité des technologies de l'information (CC). Celui-ci fournit un certain niveau d'assurance d'évaluation sur la cible d'évaluation, qui correspond au système testé, en garantissant que la cible a été évaluée dans certaines conditions, censées représenter la capacité d'un attaquant, et définies par le niveau d'assurance. Le processus est guidé par trois entités différentes.

- Un développeur qui conçoit le produit à tester.
- Un organisme de certification qui délivre le certificat sur la cible au développeur et qui est chargé de valider l'évaluation effectuée par un laboratoire de sécurité externe agréé. Il s'agit souvent d'organismes gouvernementaux, tels que l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) en France ou le Bundesamt für Sicherheit in der Informationstechnik en Allemagne.
- Un laboratoire qui effectue le test sur la cible. Les laboratoires doivent être agréés par l'organisme de certification pour délivrer le certificat. Un tel laboratoire est connu sous le nom de Centre d'Évaluation de la Sécurité des Technologies de l'Information (CESTI).

Ces acteurs font partie du processus de certification, et il est donc naturel qu'ils interviennent dans le développement de la sécurité et de la certification de l'IA. En effet, en

1. https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence

2024, l'ANSSI a publié une directive de sécurité pour l'IA générative [2]. Cependant, cette démarche est également soutenue par des chercheurs travaillant sur la sécurité théorique de l'IA et sur des implémentations sécurisées.

Cette thèse s'inscrit dans ce contexte et résulte d'une collaboration entre l'équipe TARAN de l'Institut national de recherche en informatique et en automatique (INRIA), l'université de Rennes et le CESTI Thales, dans le cadre d'une convention CIFRE. L'objectif de cette thèse est d'étudier les nouvelles attaques contre les algorithmes basés sur l'apprentissage profond en mettant l'accent sur les attaques physiques, et plus particulièrement sur les attaques par canaux auxiliaires. Cette orientation est due aux principaux acteurs de cette thèse. En effet, historiquement, le CESTI Thales s'est spécialisé dans les attaques physiques, et en complémentarité, l'équipe TARAN vise à proposer du matériel efficace et sécurisé pour les algorithmes d'apprentissage profond. Même si l'objectif final est d'améliorer la sécurité des algorithmes basés sur l'apprentissage profond, nous considérons qu'avant de proposer de nouvelles défenses, il est nécessaire de bien comprendre la portée et les cibles potentielles au sein de ces algorithmes. Par conséquent, dans cette thèse, nous nous concentrons sur le développement de nouvelles attaques, dans l'objectif d'identifier de nouveaux vecteurs d'attaque afin d'améliorer la sécurité de l'IA.

Contributions

Cette thèse se concentre sur l'amélioration des attaques existantes et l'introduction de nouvelles attaques à l'aide des attaques par canaux auxiliaires dans le but à la fois de sensibiliser à la sécurité de l'IA et de proposer des défenses adaptées. Dans cette section, nous résumons nos contributions ainsi que la structure de ce manuscrit.

Amélioration de la génération en boîte noire d'exemples adversaires via l'utilisation des attaques par canaux auxiliaires. Alors que la génération d'exemples adversaires en boîte blanche a fait l'objet de nombreuses études, leur génération dans un modèle de menace plus restrictif reste un défi à relever. Plusieurs méthodologies ont vu le jour afin d'améliorer les performances des attaques menées en boîte noire. Une solution courante consiste à utiliser les logits ou le vecteur de probabilité soit pour estimer le gradient par différence finie, soit afin d'améliorer l'entraînement d'un modèle substitut. Le problème est que l'accès à ces valeurs peut facilement être restreint soit en ajoutant des contre-mesures

2. https://cyber.gouv.fr/publications/recommandations-de-securite-pour-un-systeme-dia-generative

(bruit gaussien ou arrondi), soit en ne renvoyant que la classe prédite, c'est-à-dire celle correspondant à la probabilité maximale. Pour résoudre ce problème, nous introduisons une nouvelle attaque par canal auxiliaire afin d'extraire les logits pendant l'exécution de la fonction softmax. Dans le chapitre 5, nous décrivons notre attaque et présentons nos résultats sur un réseau de neurones intégré dans un microcontrôleur. Les résultats de ce chapitre ont été présentés lors du 16e atelier ACM sur l'intelligence artificielle et la sécurité (AISEC) en 2023.

Amélioration de la cryptanalyse de réseaux de neurones profonds par leur subdivision. Dans le cas des attaques par extraction de modèle, la menace la plus dangereuse correspond à une extraction de modèle basée sur la fidélité. En effet, dans le scénario d'une extraction de modèle fidèle réussie, l'attaquant peut accéder à une copie parfaite du modèle ciblé. Il est donc possible d'utiliser cette copie pour obtenir des avantages commerciaux, mais aussi pour lancer des attaques plus puissantes contre le réseau original dans des conditions de boîte blanche totale. Cependant, les attaques les plus performantes effectuant de telles extractions sont basées sur la cryptanalyse de réseaux et sont limitées par plusieurs facteurs. Les deux plus notables sont l'architecture, ces méthodes n'ayant été démontrées que sur des modèles entièrement connectés, et les cas particuliers de neurones, qui augmentent la complexité de l'attaque. Ces problèmes nous ont amenés à proposer un paradigme différent pour l'extraction par cryptanalyse. Alors que les attaques précédentes ciblaient le modèle dans son ensemble, nous proposons d'utiliser les attaques par canaux auxiliaires pour diviser le modèle ciblé à chaque fonction d'activation et effectuer l'attaque sur chacune des sous-parties du réseau. Cette nouvelle méthode est plus robuste aux cas particuliers de neurones et permet l'extraction d'architectures plus complexes. En particulier, nous avons pu cibler une version raccourcie de l'architecture MobileNetv1 qui ne pouvait pas être ciblée à l'aide des approches précédentes. Nous décrivons plus en détail notre méthodologie au chapitre 6.

Ces contributions soulignent la complémentarité entre les attaques logicielles et matérielles. Cependant, dans le contexte des systèmes embarqués, plusieurs contraintes semblent réduire l'efficacité des attaques logicielles. Cela conduit à la nécessité de nouvelles attaques spécifiquement conçues pour ce contexte.

Évaluation de l'impact de la quantification 8 bits sur la cryptanalyse de réseaux de neurones. Afin de déployer des réseaux de neurones profonds dans des systèmes

embarqués, plusieurs méthodes ont été mises en place pour réduire leur impact sur les ressources matérielles. L'une des plus populaires est la quantification, qui diminue le nombre de bits dédiés à la représentation des données afin de réduire la taille du modèle et de minimiser l'accès à la mémoire externe. Cependant, nous observons que la quantification des données à 8 bits réduit considérablement les performances de la cryptanalyse de réseau de neurones. Dans le chapitre 7, nous montrons que cela est dû à la quantification de l'entrée, qui introduit une petite erreur dans une partie cruciale de l'attaque : la recherche de points critiques. Pour atténuer ce problème, nous proposons une nouvelle méthode robuste à la quantification de l'entrée basée sur l'apprentissage supervisé. Nous utilisons les informations obtenues via une attaque par canal auxiliaire pour augmenter la fonction de perte durant l'apprentissage afin d'en améliorer la fidélité.

Présentation d'une nouvelle attaque permettant l'extraction de modèles robustes face à la quantification des entrées. Bien que la méthode précédente vise une extraction de modèle basée sur la fidélité, puisqu'elle repose sur l'apprentissage supervisé, elle n'offre aucune garantie de convergence vers le modèle ciblé. Ceci est très limitant, car cela implique qu'un autre attaquant pourrait reproduire l'expérience dans des conditions similaires et converger vers la cible réelle, améliorant ainsi considérablement l'extraction et les attaques potentielles ultérieures. Par conséquent, une telle attaque ne pourrait pas être directement prise en compte dans un processus d'évaluation, car elle n'offre aucune garantie de succès et, par conséquent, aucune garantie quant aux contre-mesures potentielles. Nous introduisons donc une nouvelle attaque combinant la cryptanalyse et les attaques par canaux auxiliaires. Nous utilisons d'abord la cryptanalyse sur la première couche du réseau, où l'erreur introduite par la quantification de l'entrée est minimale. Ensuite, une fois la première couche extraite, nous tirons parti de l'accès aux valeurs d'activation pour effectuer une attaque profilée sur les valeurs d'activation manipulées durant l'exécution de la fonction ReLU. En utilisant cette attaque, nous pouvons extraire les valeurs d'activation des neurones restants et construire un système basé sur des équations simples pour chaque neurone. Nous présentons la méthodologie complète derrière cette attaque dans le chapitre 8.

Avant de présenter ces contributions, nous fournissons les informations nécessaires à la compréhension de cette thèse. Au chapitre 1, nous présentons brièvement la notion essentielle d'apprentissage automatique. Ensuite, au chapitre 2, nous présentons les réseaux de neurones profonds en mettant l'accent sur leur mise en œuvre dans les systèmes em-

barqués. Nous décrivons les principes fondamentaux des attaques par canaux auxiliaires au chapitre 3. Enfin, nous détaillons les menaces qui pèsent sur l'intelligence artificielle du point de vue logiciel et matériel au chapitre 4.

Table of Contents

Introduction **1**

I Background 8

1 A Brief Introduction to Deep Learning 9

1.1 From the Perceptron to the Multi-Layer Perceptron 10
1.1.1 Mimicking the Brain with the Perceptron 10
1.1.2 Multi-Layer Perceptron and the Universal Approximation Theorem 11
1.1.3 Basic Elements and Notations in an Multi-Layer Perceptron 12
1.2 Learning Theory 14
1.2.1 Learning Paradigm 14
1.2.2 Supervised Learning Algorithms 15
1.2.3 Optimisation Challenges 18
1.2.4 Performance Evaluation and Common Datasets 22

2 Deep Neural Networks in Embedded Systems 24

2.1 Deep Neural Networks 25
2.1.1 Parametric Layers 25
2.1.2 Non-Parametric Layers 28
2.1.3 Complex Architectures 29
2.1.4 Life Cycle of Deep Neural Networks 30
2.2 Deep Neural Networks on Embedded Systems 31
2.2.1 Hardware Constraints 31
2.2.2 Layer Adaptations 32
2.2.3 Quantization 34
2.2.4 Other Compression Methods 37
2.2.5 Hardware Architectures for Embedded Deep Neural Networks . . . 38

3 Side-Channel Analysis **40**
3.1 Physical Emanations and Leakage Models . . . 41
3.1.1 Experimental Set-Up . . . 41
3.1.2 Physical Emanations . . . 42
3.1.3 Leakages Models . . . 43
3.1.4 Leakage Dimensionality Reduction . . . 44
3.2 Side-Channel Attacks . . . 46
3.2.1 Brief History and Link with Cryptosystems . . . 46
3.2.2 Simple Side-Channel Analysis . . . 46
3.2.3 Advanced Side-Channel Analysis . . . 48
3.2.4 Countermeasures . . . 54

4 Security of Deep Neural Networks **56**
4.1 Taxonomy of the Threats Against Deep Learning . . . 57
4.1.1 Valuable Assets . . . 57
4.1.2 Classification of Attacks Against Deep Learning . . . 58
4.1.3 Threat Models related to Deep Neural Networks . . . 60
4.2 Evasion Attacks . . . 61
4.2.1 Global principle . . . 61
4.2.2 White-Box Scenario . . . 62
4.2.3 Black-Box Scenario . . . 63
4.2.4 Gray-Box Scenario and Transfer-Based Attacks . . . 65
4.3 Model Extraction Attacks . . . 66
4.3.1 Principle and Objectives . . . 66
4.3.2 Accuracy-Based Model Extraction Attack . . . 69
4.3.3 Fidelity-Based Model Extraction Attack . . . 69
4.3.4 Cryptanalytical Extraction of Deep ReLU Neural Networks . . . 71
4.4 Countermeasures and Difficulties Related to Embedded Systems . . . 76
4.4.1 Countermeasures for Adversarial Examples . . . 76
4.4.2 Countermeasures for Model Extraction Attacks . . . 77
4.4.3 Natural Difficulties in Embedded Systems . . . 77

II Combining Software and Hardware Attacks to Enhance

Evasion and Model Extraction Attacks **79**

5 Improving Evasion Attacks Through Side-Channel Profiling of the softmax Function **80**

5.1 Motivations . . . 81
5.1.1 Limitations on Score-Based Attacks . . . 81
5.1.2 Side-channel Analysis as a First Block in Black-Box Attacks . . . 82
5.2 Overview of the Attack . . . 84
5.2.1 Global methodology . . . 84
5.2.2 General Observations . . . 86
5.3 Practical Experiments . . . 88
5.3.1 Threat Model . . . 88
5.3.2 Experimental Setup . . . 89
5.3.3 Analysis of the Hardware Vulnerability . . . 90
5.4 Evaluation . . . 93
5.4.1 Logit Extraction . . . 93
5.4.2 Complete Attack . . . 95
5.5 Conclusion . . . 97

6 Side-Channel Extraction of Deep-ReLU Networks with 32-bit Data **99**

6.1 Current Limitations of Fidelity-Based model Extractions in Hard-Label Settings . . . 100
6.1.1 Complexity of the Attack and Special Cases of Neurons . . . 100
6.1.2 Extracting Deep Layers . . . 102
6.1.3 Restriction to Fully-Connected-Layers . . . 103
6.1.4 Combining Cryptanalysis and Side-Channel Attacks . . . 104
6.2 Subdividing Deep Neural Networks to Improve Cryptanalytical Extraction 106
6.2.1 Splitting DNNs at Each Activation Function . . . 106
6.2.2 Improved Signature Extraction . . . 109
6.2.3 Estimation of the Sign of the Scaling Factor . . . 113
6.2.4 Choice of oracle and complete methodology . . . 115
6.3 Practical Experiments . . . 119
6.3.1 Threat Model . . . 119
6.3.2 Experimental Setup . . . 120
6.3.3 Side-Channel Analysis of the ReLU Implementation . . . 122

6.3.4 Complete Extraction of Classifiers . . . 125
6.3.5 Simulation of our Methodology in Soft-Label Setting . . . 127
6.4 Discussion and Limitations . . . 129
6.4.1 Propagation of Errors . . . 129
6.4.2 Impact of the Special-Case Neurons . . . 130
6.4.3 Extraction of the Last Layer . . . 131
6.5 Conclusion . . . 132

III Introducing New Specific Side-Channel Attack Against the Confidentiality of Deep Neural Networks 134

7 Fidelity-Based Model Extraction of Quantized Deep Neural Networks 135
7.1 Impact of Quantization on Cryptanalytical Model Extraction . . . 136
7.1.1 Simulation of the Extraction of Quantized Neurons . . . 136
7.1.2 Impact of the Input Quantization . . . 138
7.2 Active learning with a fidelity objective . . . 142
7.2.1 Moving from equalities to inequalities . . . 142
7.2.2 Global Algorithm . . . 144
7.2.3 Critical vs non-critical inequalities . . . 148
7.2.4 The Different Levels of Fidelity-Based Model Extraction Attacks . . . 149
7.3 Practical Experiments . . . 151
7.3.1 Experimental Setup and Threat Model . . . 151
7.3.2 Analysis of the Side-Channel Vulnerability . . . 152
7.3.3 Results with a Perfect Oracle Function . . . 155
7.3.4 Considering Errors in the Oracle Function . . . 157
7.4 Discussion and Perspectives . . . 160
7.4.1 Comparison with Different Threat Models . . . 160
7.4.2 Inconclusive Alternatives to Inequalities . . . 161
7.5 Conclusion . . . 162

8 Extraction of Quantized Models without a Profiling Device 164
8.1 Overview of the Attack . . . 165
8.1.1 Motivations . . . 165
8.1.2 Extraction of the First Layer . . . 167

8.1.3 Extending the Attack . . . 170
8.1.4 Complete Methodology . . . 172
8.2 Practical Experiments . . . 175
8.2.1 Threat Model and Metrics . . . 175
8.2.2 Experimental Setup and Targeted Architectures . . . 176
8.2.3 Results of the Side-Channel Attack . . . 177
8.2.4 Results of the Complete Attack . . . 181
8.3 Countermeasures . . . 188
8.3.1 Look-Up Table . . . 188
8.3.2 Shuffling the Neurons . . . 189
8.4 Conclusion . . . 191

Conclusion **193**

Bibliography **199**

List of Figures

1 Relation entre les domaines de l'apprentissage profond, de l'apprentissage automatique et de l'intelligence artificielle. . . . iii
2 Nombre de publications mentionnant l'apprentissage profond dans différents domaines sur ArXiv (Note: Le sujet, apprentissage profond et conduite autonome, par exemple, doit être explicitement mentionné dans les méta-données de l'article). . . . iv
3 Nombre de publications traitant d'une attaque contre les algorithmes basés sur l'apprentissage profond (Note: Le sujet, apprentissage profond et empoisonnement, par exemple, doit être explicitement mentionné dans les méta-données de l'article). . . . v
4 Position of Deep Learning within Artificial Intelligence. . . . 1
5 Number of publications using Deep Learning for various fields on ArXiv (Note: the subject, DL and autonomous driving, for example, must be explicitly mentioned in the metadata of the article). . . . 2
6 Number of publications of attacks on Deep Learning models on ArXiv (Note: the subject, DL and poisoning, for example, must be explicitly mentioned in the metadata of the article). . . . 3

1.1 Schematic description of a perceptron with $n = 3$. . . . 10
1.2 Output domains of the ReLU, Sigmoid and ReLU6 function on the interval [-10, 10]. . . . 11
1.3 Schematic representation of a Multi-Layer Perceptron. . . . 12
1.4 Schematic representation of the forward and backward propagation in an MLP. . . . 17
1.5 Finding the minimum of a non-convex function under different configurations with gradient descent. . . . 19
1.6 Comparison between underfitting, overfitting and a good fit for a polynomial model. . . . 20
1.7 Subsets of the MNIST, CIFAR-10 and ImageNet datsets. . . . 23

2.1 Schematic representation of a two-dimensional convolution. 25
2.2 Schematic representation of an input feature map composed of $C_l = 3$ channels with $H_l = 6$ and $W_l = 6$ on which is applied a convolutional layer with 5 filters for which $K_h = 3$ and $K_w = 3$. 26
2.3 Schematic representation of a convolution with zero-padding and a stride of 3. 27
2.4 Schematic representation of the application of a 2×2 pooling layer on a 4×4 input feature map. 28
2.5 Schematic description of a residual and a dense connection. 29
2.6 Convolutional Neural Netwok performing images classifications on ImageNet. 30
2.7 Schematic description of the depthwise separable convolution. 33
2.8 Schematic description of the quantization of the eight weights of neuron from a format using 4 bits to 3 bits. 35

3.1 Acquisition procedure of one electromagnetic trace on embedded DNN. . . 41
3.2 Schematic representation of a CMOS inverter. 42
3.3 Example of a side-channel trace and visualisation of the SNR and the PoIs. 45
3.4 Schematic representation of a side-channel trace associated with the potential processing of a naive implementation of the ReLU function. 47
3.5 Illustration of the evolution of $|\texttt{Dist}_{CPA}(\boldsymbol{T}, \boldsymbol{H})|$ over a set of simulated traces during a successful attack. 51
3.6 Illustration of the process of the k-means algorithm on a dataset of 3 classes represented by the background color. 52

4.1 Schematic description of a membership inference attack based on a set of substitute or shadow models to detect if a data is in or out out the training set as introduced in [Sho+16]. 59
4.2 Schematic description of using a poisoning attack to introduce a backdoor in a DNN. 60
4.3 Illustration of an adversarial example with FGSM [GSS14]. 62
4.4 Illustration of the difference between fidelity- and accuracy-based model extraction. 67
4.5 Neuron-induced hyperplane. 72

4.6 Illustration of the concept of dual points as introduced in [Car+25] on a Multi-Layer Perceptron (MLP) with one hidden layer of five neurons each associated with a different hyperplane represented by the colored lines. . . 75

5.1 Visualisation of the different blocks of our attack and their corresponding threat model. 83
5.2 Schematic description of the profiling and attack phases of our attack. . . . 85
5.3 Example of the distribution of logits on a real use-case over a set of $10,000$ samples. 87
5.4 Experimental setup composed of an EM probe placed on the chip of the MCU with the trigger connected to an external pin. 90
5.5 C and Assembly codes from the NNOM softmax function. 91
5.6 Preprocessing of an EM trace. 92
5.7 Success rate results on targeting NNOM softmax function. Each colour of a curve corresponds to the success rate of a given logit (out of 10) as a function of the number of traces used for the attack. 94

6.1 Visualisation of the impact of a max pooling layer on the correlation between the changes in the transition boundary and its intersection with the hyperplanes. 104
6.2 Visualisation of the impact of the subdivision on an architecture incorporating a pooling layer. 108
6.3 Illustration of the fusion of a linear layer with batch normalization. 110
6.4 Approximation of the special case neuron η as a skip connection to the neuron λ. 112
6.5 Targeted model extracted in an iterative process. 117
6.6 Extraction of the third critical point of neuron N1 of the targeted model using a binary search. 117
6.7 Signature extraction using the critical points from Stage 1. 118
6.8 Layer alignment and sign extraction using hypothesis testing. 118
6.9 Evolution of the success rate of the extraction of the state of a neuron as a function of the number of side-channel traces. 124
6.10 Comparison of the proportion of requests dedicated to normal versus special-case neurons. 131

7.1 Different targets for quantization in our experiments. 137
7.2 Simulation of the impact of full quantization on the extraction of a perceptron for different quantization levels. 138
7.3 Simulation of the estimation of the critical points for different quantization levels. 139
7.4 Schematic representation of the impact of input quantization on the search for critical points. 140
7.5 Simulation of the estimation of the critical points for different input size with quantization at 8-bits. 141
7.6 Impact of the input size on the extraction of the weights of 8-bit fully quantized perceptrons. 142
7.7 Schematic representation of moving from critical equations to critical inequalities. 144
7.8 Schematic representation of our first step on an MLP with two hidden layers. 145
7.9 Assembly code from the NNOM ReLU function. 153
7.10 SNR associated with the first 10 neurons in one layer of 64 neurons. 154
7.11 Impact of errors in the side-channel attacks on our extraction methodology on the example of a VGG11 model. 158

8.1 Impact of moving from critical equations to random equations to estimate the weights of a neuron. 168
8.2 Results of our simulated extraction of the first layer of quantized MLP using the NNoM framework. 169
8.3 First step of our methodology performing cryptanalytical extraction of the first layer. 172
8.4 Filtering our the incorrect signature based on the state of the critical point in the second stage of our framework. 173
8.5 Profiling one neuron on the targeted device to extract the activation values for different scaling factor values. 173
8.6 Schematic representation of the extraction of the scaling factor by selecting the model leading to the best results. 174
8.7 Illustration of Stage 5 corresponding to the iterative extraction of the remaining neurons. 175
8.8 SNR on the activation value on the complete profiling set. 178

8.9 Evolution of the SNR on the activity of the neuron for the active and inactive subgroups. 179
8.10 Evolution of the rank for each possible hypothesis in function of the number of attack traces. 180
8.11 Statistical analysis of the attack set corresponding to neurons 5 and 55. . . 181
8.12 Evolution of the percentage of successfully extracted neurons with the RANSAC algorithms for different initial probabilities. 183
8.13 Evolution of the number of queries to achieve a perfect extraction of all 64 neurons using the updated Random Sample Consensus (RANSAC) algorithm. 184
8.14 Evolution of the number of iterations of the RANSAC algorithm for different configurations. 187
8.15 Comparison between the t-test on the state of the neuron between a basic implementation and our implementation with a look-up table. 189
8.16 Evolution of the t-statistic on the activity of the neuron for different layer sizes with our shuffling countermeasure. 191

List of Tables

5.1 Preprocessing on the EM traces depending on the obtained SNR value. . . 92
5.2 Comparison between our methodology and generation on a substitute network. 96

6.1 Overview of the state-of-the-art of fidelity-based model extraction attacks. 119
6.2 Description of the architecture of the shortened MobileNetv1. 120
6.3 Description of a depth-wise separable convolution taking N_c channel as input. 121
6.4 Description of the architecture of the large MLP. 121
6.5 Summary of the targeted architectures. 121
6.6 Comparison in hard-label settings. 125
6.7 Maximum error on the weights with and without the residual error on the MLP architecture from [Car+25] 126
6.8 Comparison of DNNs trained for regression in soft-label settings with 64-bit data. 129
6.9 Results of our methodology against targeted architectures by SOTA methodologies using 32-bit data. 129
6.10 Layer-wise statistics of the extraction of the shortened MobileNetv1. 130

7.1 Description of the architecture of the MLP. 151
7.2 Description of the architecture of the CNN. 152
7.3 Evaluation of our methodology on simple architectures. 155
7.4 Evaluation of our methodology on state-of-the-art architectures. 157
7.5 Evaluation of our methodology using a single activation value per neuron. . 159
7.6 Comparison with [Hec+23]. 161
7.7 Comparison with [Rak+21]. 161

8.1 Evaluation of the performance of our attack on MLPs quantized to 8 bits using the NNoM framework [Ma20]. 184

List of Acronyms

AI	Artificial Intelligence
ANSSI	Agence Nationale de la Sécurité des Systèmes d'Information
ASCA	Advanced Side-Channel Analysis
ASIC	Application-Specific Integrated Circuits
AuA	Accuracy under Attack
BCE	Binary Cross-Entropy
BN	Batch Normalization
BNN	Binarized Neural Network
CC	Common Criteria for Information Technology Security Evaluation
CEMA	Correlation ElectroMagnetic Analysis
CMOS	Complementary Metal Oxide Semiconductor
CNN	Convolutional Neural Network
CPA	Correlation Power Analysis
DEMA	Differential Electromagnetic Analysis
DL	Deep Learning
DLSCA	Deep Learning Side-Channel Attack
DNN	Deep Neural Network
EAL	Evaluation Assurance Level
EM	ElectroMagnetic
FGSM	Fast Gradient Sign Method
FPGA	Field-Programmable Gate Array
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
INRIA	Institut National de Recherche en Informatique et en Automatique

IP	Intellectual Property
ITSEF	Information Technology Security Evaluation Facility
LASSO	Least Absolute Shrinkage and Selection Operator
LSTSQ	Least Squares
MCU	Micro-Controller Unit
MLaaS	Machine-Learning-as-a-Service
MBGD	Mini-Batch Gradient Descent
ML	Machine Learning
MLP	Multi-Layer Perceptron
MLR	Multinomial Logistic Regression
MSB	Most Significant Bit
MSE	Mean-Squared Error
NAS	Neural Architecture Search
PoI	Points of Interest
PTQ	Post Training Quantization
QAT	Quantization Aware Training
RANSAC	Random Sample Consensus
ReLU	Rectified Linear Unit
SCA	Side-Channel Analysis
SSCA	Simple Side-Channel Analysis
SNR	Signal-to-Noise Ratio
SOTA	State-Of-The-Art
SGD	Stochastic Gradient Descent
TOE	Target of Evaluation
TPU	Tensor Processing Unit
XOR	Exclusive-Or

NOTATIONS

Sets and Variables

$\mathcal{X}$	A set
$\mathbb{Z}$	The set of integers
$\mathbb{N}$	The set of natural integers
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^+$	The set of positive real numbers
$\mathbb{F}_2^n$	The set of binary values composed of n bits
$\mathcal{M}_{n,d}(\mathbb{R})$	The set of matrix composed of n rows and d columns
$\{0,1\}$	The set containing 0 and 1
$[a,b]$	The real interval between, and including, a and b
$[\![0;n]\!]$	The interval composed of the integers between, and including, a and b
$\mathbf{X}$	A tensor (vector, matrix, *etc.*)
$\mathbf{X}_i$	i-th element of tensor $\mathbf{X}$
$\mathbf{X*}$	The adversarial example associated to input $\mathbf{X}$
$D_{\boldsymbol{X},\boldsymbol{Y}}$	A dataset composed of the tensors $\boldsymbol{X}$ as inputs and $\boldsymbol{Y}$ as labels

Functions and Linear Algebra Operations

$f(\cdot)$	A function
$f_\theta(\cdot)$	f a parametric function of θ
$f \circ g$	The composition of function f and g
$\odot$	Hadamard product
$\|\| \cdot \|\|_p$	l_p norm
$\nabla_x f$	Gradient of f with respect to x

Distribution and Probabilities

$\texttt{Pr}[X]$	Probability of observing an event X
$\texttt{Pr}[Y\|X]$	Conditional probability of Y given X
$\mathcal{D}_{\mathcal{X}}$	Distribution over the set of data $\mathcal{X}$
$\mathbb{E}[X]$	Expected value of the random variable X
$\mathbb{V}[X]$	Variance of the random variable X
$\mathcal{N}_D(\boldsymbol{\mu}, \sigma)$	Normal distribution of mean μ and variance σ
$\binom{n}{k}$	Binomial coefficient

Introduction

Over the past decade, Deep Learning (DL) algorithms have begun to emerge in a wide variety of domains. These algorithms belong to the field of Artificial Intelligence (AI) as they aim at simulating intelligence. Furthermore, since they automatically learn to solve a task by interacting with and identifying patterns in a dataset, they correspond more precisely to Machine Learning (ML) algorithms. DL, therefore, represents a subfield of ML, itself being part of AI, as described in Figure 4.

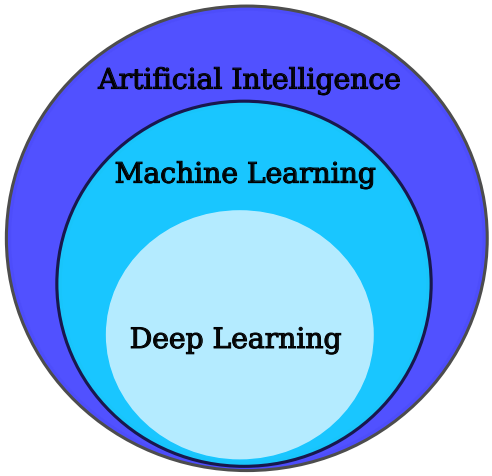


Figure 4 – Position of Deep Learning within Artificial Intelligence.

The specificity of DL lies in its inspiration and purpose to mimic brain functions by creating artificial neural networks composed of a large number of perceptrons organized in networks emulating the neurons and the synapses of the brain. While the first work on this subject can be dated to 1958, with Rosenblatt *et al.* [Ros58] describing the structure of the perceptron, DL only gained in popularity recently. Indeed, the first algorithms were limited in their capacity to learn complex functions, reducing their interest in real-world datasets. However, the combined improvement made from the retropropagation algorithm [RHW86], along with the increase in both data availability and computational power from the use of Graphical Processing Units (GPUs), allowed DL to break through. In particular, in 2012, Alex Krizhevsky *et al.* [KSH12] reduced the *top-5* error on the ImageNet dataset [Den+09] to 15.3%, which was 10.8% better than the second-best algorithm at that time,

starting the era of DL algorithms. Since then, these algorithms have become standard in domains as different as autonomous driving, cancer detection, or malware classification. This can clearly be observed by the increased number of publications mentioning DL in their abstract within these fields, as shown in Figure 5.

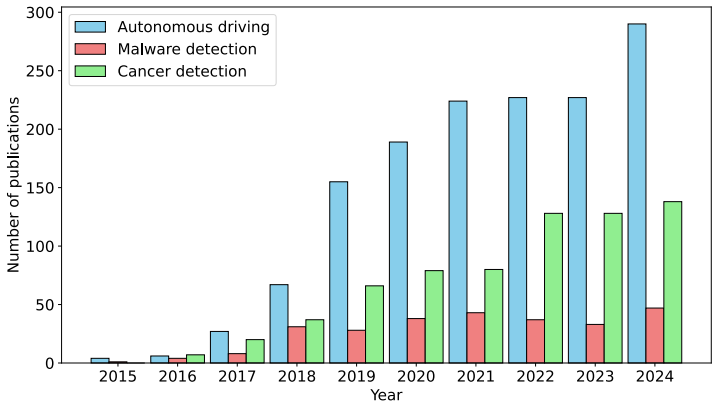


Figure 5 – Number of publications using Deep Learning for various fields on ArXiv (Note: the subject, DL and autonomous driving, for example, must be explicitly mentioned in the metadata of the article).

However, with this success, the Intellectual Property (IP) concerning DL algorithms and their data became a critical subject, leading to various concerns regarding the security of these algorithms.

Indeed, the increased complexity of the tasks, combined with the time and resources needed to train a Deep Neural Network (DNN), has transformed trained models into valuable assets. As a consequence, model extraction attacks, *i.e.*, attacks aiming for the extraction of DNNs to acquire a model without having to perform the training process, emerged, as well as attacks trying to retrieve the training data. Another threat to consider is that an attacker could choose to target the decision process of the AI. In the critical domains that we have previously mentioned, this could have disastrous consequences. In the past few years, numerous attacks of this type against DNNs have been proposed by researchers, as represented in Figure 6.

The most popular and widely studied attack against DNNs corresponds to evasion attacks [GSS14; CW16], where an attacker crafts an input that is misclassified by the targeted DNN while minimizing the changes made to the original input. While this attack occurs at inference time, the DNN is also vulnerable during its training. It is possible to introduce bias in the data, *i.e.*, to poison the data, thereby reducing the performance of

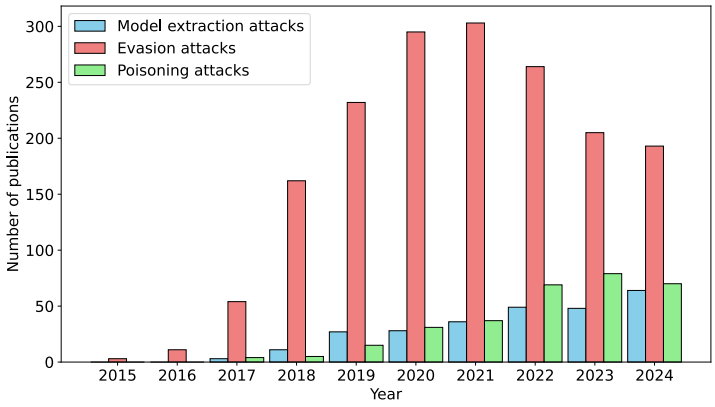


Figure 6 – Number of publications of attacks on Deep Learning models on ArXiv (Note: the subject, DL and poisoning, for example, must be explicitly mentioned in the metadata of the article).

the DNN and potentially introducing backdoors. This highlights the duality of the threats against DNNs. On one hand, one should consider the attack threatening the confidentiality of the model and its data. On the other hand, it is also critical to view the DNN as part of a whole and deal with the attacks that try to perturb the decision process of the DNN.

Finally, similarly to cryptographic components, the DNN must not be seen as an algorithm but also as an implementation that will run on a physical device. This implementation can be flawed or tampered with, leading to different types of attacks, such as physical attacks. This type of attack uses physical properties to infer some information on the processing of the algorithm in a passive way, or can even be employed to modify the nominal behavior of the algorithms. While these attacks have been studied on cryptographic systems for the past twenty years [Koc96; KJJ99; BCO04], it is only recently that they have been applied to DNNs [Wei+18; Bat+19]. Since, numerous attacks threatening various aspects of AI security, *e.g.*, the confidentiality of the model or data [Hec+23; Jou+23b], the integrity of the prediction [RHF19], have emerged.

However, despite all these concerns for the security of AI, the performance of these algorithms, alongside their growing popularity, has led to their widespread adoption. Consequently, it is necessary for governments and industries to provide some guidelines to evaluate the security of products, including AI-based solutions. One of the first steps was the AI Act voted in 2024 by the European deputies [3]. The next step is to propose a set of

3. https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence

rules to certify the security of AI products. In cybersecurity, this set of rules for certification is defined by the certification schemes and concerns all parts of cybersecurity, *e.g.*, side-channel attacks, cryptography, or software attacks. The most common scheme is the Common Criteria for Information Technology Security Evaluation (CC) [4], which provides a certain Evaluation Assurance Level (EAL) on the Target of Evaluation (TOE), which corresponds to the system tested, by ensuring that the TOE has been evaluated in certain conditions, supposed to represent the capacity of an attacker, and defined by the EAL. The process is guided by three different entities:

- A developer who designs the product to be tested.
- A certification body that delivers the certificate on the TOE to the developer and is in charge of validating the evaluation performed by an external licensed security laboratory. They are often governmental organizations, such as the Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) in France or the Bundesamt für Sicherheit in der Informationstechnik in Germany.
- A laboratory that performs the test of the TOE. The laboratories must be licensed by the certification body to issue the certificate. Such a laboratory is known as an Information Technology Security Evaluation Facility (ITSEF).

These actors are part of the certification process in cybersecurity, and it is therefore natural that they act in the development of the security and the certification of AI. Indeed, in 2024, ANSSI published a security guideline for generative AI [5]. However, this process is also supported by researchers working on the theoretical security of AI and on secure implementations.

This thesis originates from this context, resulting in a collaboration between the TARAN team from the Institut National de Recherche en Informatique et en Automatique (INRIA), Rennes University, and the Thales ITSEF, leading to a thesis under a CIFRE convention. The purpose of this thesis is to study new attacks on DL algorithms with an emphasis on physical attacks, and more specifically, on side-channel attacks. This focus is due to the actor of this thesis. Indeed, historically, the Thales ITSEF has specialized in physical attacks, and complementarily, the TARAN team aims to propose efficient and secure hardware for DL algorithms. In this thesis, we focus on developing new attacks, as we believe that identifying new attack vectors is the first step toward improving the security of a product. Therefore, we argue that before proposing new defenses, it is

4. https://www.commoncriteriaportal.org/files/ccfiles/CEM2022R1.pdf
5. https://cyber.gouv.fr/publications/recommandations-de-securite-pour-un-systeme-dia-generative

necessary to fully understand the scope and the potential assets that can be targeted with AI.

Contributions and Thesis Outline

This thesis focuses on improving existing attacks and introducing new ones by using Side-Channel Analysis (SCA) for the purpose of both increasing awareness regarding AI security and proposing adapted defenses. In this section, we summarize our contributions and describe the outline of this manuscript.

Combining software and SCA to improve black-box generation of adversarial examples. While the generation of powerful **adversarial examples** in white-box settings has been vastly studied, crafting such inputs in a more restricted threat model remains an open challenge. Several attacks have emerged to improve the performance of attacks made in **black-box** settings, while trying to reduce the level of prior knowledge of the attacker compared to white-box attacks. One common solution is to **use the logits** or **the probability vector** to either estimate the gradient through finite difference or improve the training of a substitute model. The issue is that access to these values can easily be restricted either by adding countermeasures (Gaussian noise or rounding) or by only returning the hard-label, *i.e.*, the label corresponding to the maximum probability. To deal with this obstacle, we introduce a new **side-channel attack to extract the logits** during the processing of the softmax function. In Chapter 5, we describe our attack and present our results on a DNN embedded on an Micro-Controller Unit (MCU). The results of this chapter have been presented at the 16th ACM Workshop on Artificial Intelligence and Security (AISEC) in 2023.

Splitting DNNs with SCA to enhance cryptanalytical extraction. In the case of model extraction attacks, the most dangerous threat corresponds to a **fidelity-based model extraction**. Indeed, in the scenario of a successful fidelity-based model extraction, the attacker can access to a copy of the targeted DNN. Therefore, it is possible to use the extracted model to achieve competitive advantages but also to mount more powerful attacks on the target in complete white-box settings. However, attacks performing such extractions are based on **cryptanalytical extraction** of DNNs and are limited by several factors. The two most notable are the architecture, these methods have only been demon-

strated on **fully connected** models, and the **special cases of neurons**, which increase the complexity of the attack. These issues have led us to propose a different paradigm for cryptanalytical extraction. While previous attacks target the DNN as a whole, we propose to use **SCA to split the targeted model at each activation function** and perform cryptanalytical extraction on the subparts of the network. This new method is more **robust to special cases of neurons and allows for the extraction of more complex architecture**. In particular, we were able to target a shortened version of the **MobileNetv1**. Architecture that could not be targeted using State-Of-The-Art (SOTA) approaches. We further describe our methodology in Chapter 6.

These contributions highlight the complementarity between software and hardware-based attacks. However, in the context of embedded systems, several constraints appear to reduce the efficiency of software-based attacks. This leads to the need for new attacks specifically designed for this context.

Assessing the impact of 8-bits quantization on cryptanalytical extraction. To deploy DNNs in **embedded systems** several methods have been introduced to reduce their impact on the resources. One of the most popular is **quantization**, which diminishes the number of bits dedicated to the data representation to decrease the model size and minimize off-chip memory access. We observe that quantizing to 8**-bit data** drastically decreases the performance of **cryptanalytical extraction**. In Chapter 7, we further demonstrate that this is due to the **input quantization** which introduces a small error in a crucial part of the cryptanalytical extraction: **the critical point search**. To mitigate this issue, we propose a new method robust to input quantization based on **supervised learning**. We use side-channel information to augment the traditional loss on the hard-label in order to improve the fidelity of the learning process.

Introducing a new model extraction attack robust to input quantization. While the previous method aims for a **fidelity-based model extraction**, since it is based on supervised learning, it **offers no guarantee of convergence to the targeted model**. This is critical as it implies that another attacker could reproduce the experiment and converge to the true target, significantly improving the extraction and potential further attacks. Therefore, such an attack could not be directly considered in an evaluation process, as it offers no guarantee of the success of the attack and, consequently, of the potential

countermeasures. As a consequence, we introduce a **new attack combining cryptanalytical extraction** and **SCA**. We use cryptanalytical extraction on the first layer, where the error introduced by the input quantization is at its minimum. Once the first layer is extracted, we take advantage of having access to the activation values to profile the side-channel observables associated with the processing of the Rectified Linear Unit (ReLU) function. Based on this, we can extract the activation values of the remaining neurons and construct a system based on simple equations for each neuron. We present the complete methodology behind this attack in Chapter 8.

Before presenting these contributions, we provide the necessary background for the comprehension of this thesis. In Chapter 1, we briefly introduce the essential notion of ML. Then, in Chapter 2, we present DNNs with a focus on their implementation in embedded systems. We describe the fundamental principles of SCA in Chapter 3. And finally, we detail the threats against AI from both a software and hardware perspective in Chapter 4.

Part I

Background

Chapter 1

A Brief Introduction to Deep Learning

The purpose of DL algorithms is to solve a given task automatically by first learning to approximate a function on a training set and then using this function to predict outcomes on new data. This method achieves spectacular performance and even outperforms humans on complex tasks such as image classification or playing chess. To obtain such results, complex models must be carefully designed and trained. However, modern architectures all stem from the same building blocks and share some common principles that we need to outline before moving on to more complex architectures.

In this chapter, we present the necessary principles of DL to the comprehension of this thesis. We will first introduce the basic block of the architecture, the perceptron, and then describe how such a model can be improved to represent any function by combining it into more complex models: the MLP. Finally, we will conclude by describing the different learning paradigms and algorithms that allow ML algorithms to automatically learn such a function with a MLP and a dataset.

Contents

1.1 From the Perceptron to the Multi-Layer Perceptron 10

1.1.1 Mimicking the Brain with the Perceptron 10

1.1.2 Multi-Layer Perceptron and the Universal Approximation Theorem 11

1.1.3 Basic Elements and Notations in an Multi-Layer Perceptron . . 12

1.2 Learning Theory 14

1.2.1 Learning Paradigm 14

1.2.2 Supervised Learning Algorithms 15

1.2.3 Optimisation Challenges 18

1.2.4 Performance Evaluation and Common Datasets 22

1.1 From the Perceptron to the Multi-Layer Perceptron

1.1.1 Mimicking the Brain with the Perceptron

In 1958, Franck Rosenblatt introduced what would become the basic block of DL: the *perceptron* [Ros58]. We propose a graphical representation of the perceptron in Figure 1.1, with $\boldsymbol{X} \in \mathbb{R}^n$ the **input vector** of the perceptron, $n \in \mathbb{N}$ the dimension of the input space,

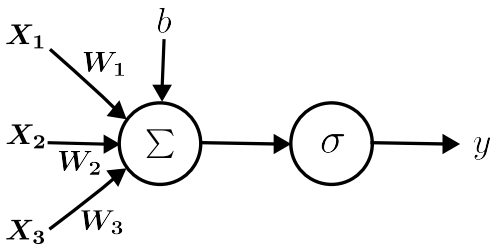


Figure 1.1 – Schematic description of a perceptron with $n = 3$.

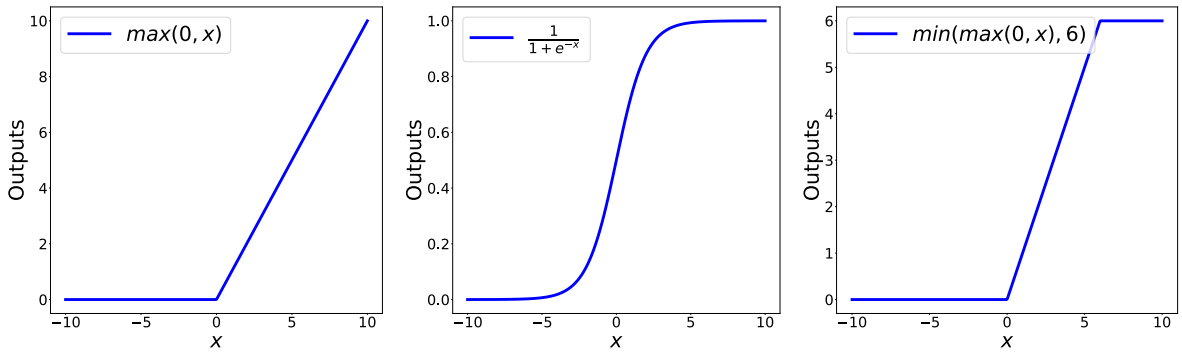
and an **output** $y \in \mathbb{R}$. The following elements can fully characterize a perceptron:

- a set of **weights** denoted $\boldsymbol{W} \in \mathbb{R}^n$,
- a **bias** $b \in \mathbb{R}$, and
- an **activation function** $\sigma : \mathbb{R} \to \mathbb{R}$.

The output y of the perceptron is given by

$$y = \sigma(\boldsymbol{X}^T \boldsymbol{W} + b) = \sigma(\sum_{i=1}^{n} \boldsymbol{X_i} \cdot \boldsymbol{W_i} + b) \tag{1.1}$$

As we can see, without the activation function, the perceptron corresponds to a linear transformation. Therefore, these are usually non-linear functions. For example, the one used in the original article [Ros58] corresponds to a threshold function. This construction was designed to mimic the brain and how an electrical signal activates neurons. Since a large variety of activation functions have been introduced, we can cite, for example, the ReLU [NH10], the Sigmoid [RHW86] or the hyperbolic tangent (Tanh) [Lec89], and their following variations, such as Softmax or ReLU6 [Kri10], which are derived respectively from the Sigmoid and the ReLU function. We provide in Figure 1.2 three graphs corresponding to the mapping of the activation functions that we will use in this thesis: ReLU, Sigmoid, and ReLU6.



(a) ReLU function. (b) Sigmoid function. (c) ReLU6 function.

Figure 1.2 – Output domains of the ReLU, Sigmoid and ReLU6 function on the interval [-10, 10].

Despite proving its ability to represent specific functions, the perceptron is not appropriate for tasks requiring a nonlinear separation, such as approximating the Exclusive-Or (XOR) function. To represent these more complex functions, one solution is to combine perceptrons, or as they are referred to in this case, neurons, and create a more complex architecture by stacking them vertically and horizontally in several layers. Such architectures correspond to MLPs.

1.1.2 Multi-Layer Perceptron and the Universal Approximation Theorem

The genesis of the MLP can be traced to the Universal Approximation Theorem. In 1989, Cybenko [Cyb89] proved that by stacking vertically a finite number of neurons using a sigmoidal function, it was possible to approximate any continuous function of real variables on a closed and bounded subset of $\mathbb{R}^n$. In particular, we have:

Theorem 1.1 ([Cyb89]) *Let $f_t : \mathbb{R}^d \to \mathbb{R}$ be a continuous function and $\sigma : \mathbb{R} \to \mathbb{R}$ a sigmoidal activation function. Then, for any $\epsilon > 0$, there exits two vectors $\boldsymbol{\alpha}$, $\boldsymbol{b} \in \mathbb{R}^n$ with $n \in \mathbb{N}$ and a matrix $\boldsymbol{W} \in \mathcal{M}_{d,n}(\mathbb{R})$ such that:*

$$|\sum_{i=1}^{n} \boldsymbol{\alpha_i} \sigma(\boldsymbol{X}^T \boldsymbol{W_i} + \boldsymbol{b_i}) - f_t(\boldsymbol{X})| < \epsilon \tag{1.2}$$

This increased the interest in MLPs, as it provides a theoretical guarantee that even a single-layer MLP is enough to represent a large range of functions under the hypothesis

that the number of neurons is sufficiently large. Finally, MLPs were fully justified when it was proved that increasing the depth of the model offered more advantages in terms of capacity to approximate complex functions than increasing the width [Mon+14].

1.1.3 Basic Elements and Notations in an Multi-Layer Perceptron

In an MLP, neurons are stacked both horizontally between different layers and vertically within each layer. Each neuron is connected to all neurons of the previous layer, forming what we describe as a *fully connected* or *dense* layer as represented in Figure 1.3.

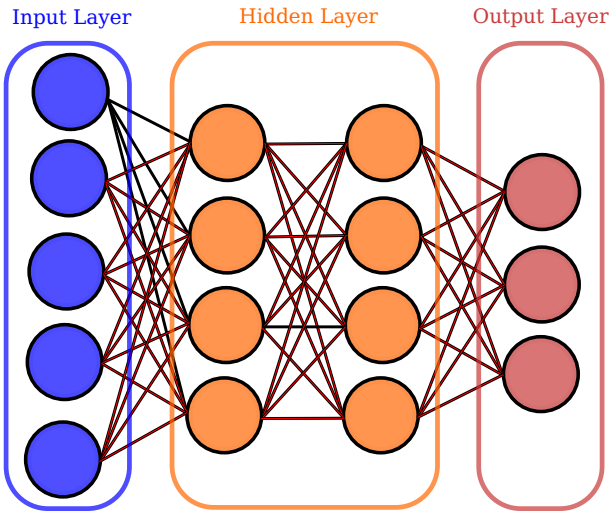


Figure 1.3 – Schematic representation of a Multi-Layer Perceptron.

Traditionally, we consider three different kinds of layers for an MLP: the input layer, the hidden layers, and the output layer. The output layer is highly dependent on the task for which the MLP is designed. In this thesis, we restrict ourselves to only two tasks: *regression* and *classification*.

Definition 1.1 (Regression Task) *In a regression task, the purpose of the model is to predict a continuous value.*

A basic example corresponds to predicting the temperature for the next day given a set of measurements. Therefore, for an MLP designed for a regression task, the last layer is composed of only one unit, which outputs a continuous value, in this case, the temperature.

Definition 1.2 (Classification Task) *For a classification task, the model is designed to predict a specific class among a set of possible classes and therefore, the output of the model can be seen as discrete.*

In the same context, an example of a classification task would correspond to estimating the global weather, *i.e.*, sunny, rainy, *etc.*, given the same data. This task is usually achieved by estimating the conditional probability of each class given the data using the model and selecting the class with the highest probability. In this scenario, the final layer is connected to several neurons, each corresponding to one class or label, *i.e.*, one of the possible outputs, and the prediction corresponds to the neuron with the highest value. Figure 1.3 represents an MLP designed for a classification between three possible labels.

While the basic block of the MLP is the neuron, these share the same components, *parameters* (bias and weights), and activation function as the perceptron. However, each neuron has different parameters, and therefore, we describe the weights matrix of the layer l of the MLP with the following notation $\boldsymbol{W_l} \in \mathbb{R}^{n,m}$ with n (resp. m) the number of neurons in the preceding (resp. current) layer and $\boldsymbol{b_l} \in \mathbb{R}^m$ the bias vector of the layer l. When there is no confusion possible, we will continue to use the notation $\boldsymbol{W}$ to describe the weights associated with a single neuron, similarly to Equation 1.1.

An MLP will, therefore, process an input sequentially, layer by layer, until the last layer, and produce an output. More formally, we use the following definition for the MLP:

Definition 1.3 (MLP) *Given an input $\boldsymbol{X} \in \mathcal{X}$, the MLP corresponds to a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ parametrized by a set of parameters $\boldsymbol{\theta} \in \Theta$ which can be defined as a composition of L functions, corresponding to the number of layers, alternating between a linear function f_l and a component-wise activation function σ_l:*

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}) = \sigma^L \circ f_L \circ \sigma_{L-1} \ldots \circ \sigma_1 \circ f_1 \circ \boldsymbol{X} \tag{1.3}$$

The number of layers L is also referred to as the *depth* of the MLP. Usually, the activation function remains the same throughout the MLP, except in the last layer, where it is common to remove this function for regression tasks or include a normalization step for classification. This normalization is done by applying the Sigmoid (resp. Softmax) function for binary (resp. multiclass) classification on the output of the last layer f_L.

Definition 1.4 (Logits vector) *Let $\boldsymbol{X} \in \mathcal{X}$ be an input to the MLP defined by the function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ as in Definition 1.3. The logit vector $\boldsymbol{Z}(\boldsymbol{X})$ corresponds to the output of the function $Z : \mathcal{X} \to \mathcal{L}$. In practice we usually have $\mathcal{L} \subseteq \mathbb{R}^{|\mathcal{Y}|}$, and this function associates an input $\boldsymbol{X}$ to the last layer:*

$$\boldsymbol{Z}(\boldsymbol{X}) = f_L \circ \sigma_{L-1} \ldots \circ \sigma_1 \circ f_1 \circ \boldsymbol{X}. \tag{1.4}$$

When there is no confusion, we also use the notation f to represent directly the parametric function $f_{\boldsymbol{\theta}}$ and f_l to represent the composition of activation functions and affine transformations from the first layer to layer l. After normalizing the logits vector $\boldsymbol{Z}(\boldsymbol{X})$, the output of the MLP corresponds to a probability vector, and its entries are the *confidence scores*, also referred to as the *soft-labels*. As previously mentioned for classification tasks, the predicted class $y \in \{1, ..., |\mathcal{Y}|\}$ corresponds to the entry with the maximum value. Therefore, $y = \texttt{argmax}(f(\boldsymbol{X}))$. In analogy with the soft-labels, which correspond to the probabilities, the predicted class is also referred to as the *hard-label*.

While the Universal Approximation Theorem offers some theoretical guarantees on the capacity of MLPs to approximate any bounded and regular function, in practice, the search for the optimal parameters remains a difficult process.

1.2 Learning Theory

1.2.1 Learning Paradigm

The principle of ML, and consequently of DL, is to automatically learn from a set of data the parameters of a model. Once this is done, the model parameters are frozen, and it can be used as is to make predictions on new unseen data. This two-phase process is central to ML. The learning phase corresponds to the *training* phase, and the second one is the *inference* phase. So far in this thesis, we have only presented MLPs through the prism of their capacity to approximate complex functions, limiting us only to the inference phase. However, as previously mentioned, the performance of the algorithm is conditioned by the training phase and its convergence toward the optimal parameters, making this step critical. While in ML the parameters are always automatically learned from data, there are several learning paradigms for different contexts. The two main ones are *unsupervised* and *supervised* learning.

Definition 1.5 (Unsupervised Learning) *Unsupervised learning corresponds to a paradigm where a learning algorithm uses a collection of data to learn the properties within the dataset. In particular, the goal is usually to learn the probability distribution that generated the dataset. Unsupervised learning can be naturally applied to clustering or data reduction tasks.*

Definition 1.6 (Supervised Learning) *In supervised learning, the algorithm has access to a dataset $D_{\boldsymbol{X},\boldsymbol{Y}}$, composed of a set of inputs $\boldsymbol{X} \in \mathcal{X}$, along with their associated labels $\boldsymbol{Y} \in \mathcal{Y}$, and aims at learning $\Pr[\boldsymbol{Y}|\boldsymbol{X}]$ to then predict $\boldsymbol{Y}$ from a new unseen $\boldsymbol{X}$. Supervised learning is usually associated with tasks such as regression and classification.*

Unsupervised learning offers the advantage of not requiring the dataset to be labeled, which can be difficult for some tasks. While several other learning paradigms, such as **semi-supervised learning** or **reinforcement learning**, have been introduced, in this thesis, we will only consider DL algorithms trained via supervised learning.

1.2.2 Supervised Learning Algorithms

As previously mentioned in a supervised learning context, the algorithm has access to a training dataset $D_{\boldsymbol{X},\boldsymbol{Y}}$, composed of a set of inputs and their corresponding labels $(\boldsymbol{X}, \boldsymbol{Y}) \in \mathcal{X} \times \mathcal{Y}$. The purpose of such an algorithm is to learn a parametric function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ that can predict the true value of $\boldsymbol{Y}$ from $\boldsymbol{X}$ by using this training dataset. To quantify the performance of this function, we use a `loss` function, $\texttt{loss} : \Theta \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$, that computes the difference or the error between the prediction and the expected value. The goal of the *learning algorithm* is to optimize the parameters of the function f to minimize the loss function over the training dataset. It is therefore natural to introduce a *global loss* function $\mathcal{L}$ of the parameters $\boldsymbol{\theta}$ averaged over the set of inputs composed of N training points:

$$\begin{aligned} &\mathcal{L} : \Theta \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+, \\ &(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) \mapsto \frac{1}{N} \sum_{j=1}^{N} \texttt{loss}(f_{\boldsymbol{\theta}}(\boldsymbol{X_j}), \boldsymbol{Y_j}). \end{aligned} \tag{1.5}$$

with Θ the set of all possible parameters for the function $f_{\boldsymbol{\theta}}$.

In practice, several loss functions can be used depending on the type of task. For regression tasks, the most common one is the Mean-Squared Error (MSE):

Definition 1.7 (MSE Loss function) *Let $\boldsymbol{X} \in \mathcal{X}$ be a set of inputs to the model defined by the parametric function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ and $\boldsymbol{Y} \in \mathcal{Y}$ be their corresponding labels. The MSE loss function is given by*

$$\texttt{loss}_{MSE}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) = \left(f_{\boldsymbol{\theta}}(\boldsymbol{X}) - \boldsymbol{Y}\right)^2. \tag{1.6}$$

For binary classification tasks, one popular choice is the Binary Cross-Entropy (BCE) loss. In this case, the output of the model must be a probability. Therefore, the last activation function of the model is usually the sigmoid function. This loss takes advantage of the fact that the labels are encoded as binary values, *i.e.*, 0 or 1, and computes the loss accordingly.

Definition 1.8 (BCE Loss function) *Let $\boldsymbol{X} \in \mathcal{X}$ be a set of inputs to the model defined by the parametric function $f_{\boldsymbol{\theta}} : \mathcal{X} \to [0, 1]$ and $\boldsymbol{Y} \in \{0, 1\}^N$ be their corresponding labels. The BCE loss function is given by*

$$\texttt{loss}_{BCE}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) = -\left(\boldsymbol{Y}\,\texttt{log}(f_{\boldsymbol{\theta}}(\boldsymbol{X})) + (1 - \boldsymbol{Y})\,\texttt{log}\,(1 - f_{\boldsymbol{\theta}}(\boldsymbol{X}))\right) \tag{1.7}$$

For multiclass classification problems, the BCE loss can be adapted to become the cross-entropy loss.

Due to the non-linearity introduced by the activation functions, these loss functions become non-convex, and the minimization of the global loss function $\mathcal{L}$ is usually performed by solving the optimization problem in an iterative process. The most classical approach corresponds to the *gradient descent* [CAU47]. At each iteration i, the parameters of the model are updated using the gradient of the global loss function with respect to the parameters $\nabla_{\boldsymbol{\theta_i}}\mathcal{L}(\boldsymbol{\theta_i}, \boldsymbol{X}, \boldsymbol{Y})$ such as

$$\boldsymbol{\theta_{i+1}} = \boldsymbol{\theta_i} - \eta\nabla_{\boldsymbol{\theta_i}}\mathcal{L}(\boldsymbol{\theta_i}, \boldsymbol{X}, \boldsymbol{Y}) \tag{1.8}$$

with $\eta \in \mathbb{R}^+$ the *learning rate*, a training hyperparameter modulating the magnitude of the update of the parameters at each iteration.

While this strategy allows for updates to the parameters, it requires an efficient method for estimating the gradient of the model at each iteration. This computation can be achieved through *backpropagation*. Backpropagation is an algorithm proposed in 1986

by Rumelhart [RHW86], which corresponds to a new method to compute the partial derivative of an MLP using the chain rule. It takes advantage of the forward propagation, where the inputs are processed layer-by-layer onward until they reach the loss function, which produces a scalar by creating a new pass that flows backward. The intuition behind the backpropagation is to use the *chain rule* to peel off the layers of the MLP and compute the gradient for the parameters of each layer. Indeed, the chain rule states that for two vectors $a \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$ and two functions $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$ such that $b = f(a)$ and $g(b) = g \circ f(a)$, then:

$$\nabla_a g(f(a)) = \boldsymbol{J}_g(a)^T \nabla_{g(a)} f(g(a)) \tag{1.9}$$

with $\boldsymbol{J}_g(a)$ the Jacobian matrix of g at the point a.

Using the chain rule, it is therefore possible to move backward and compute the gradient with respect to the parameters, starting from the last layer and backpropagating until the first layer, as illustrated in Figure 1.4.

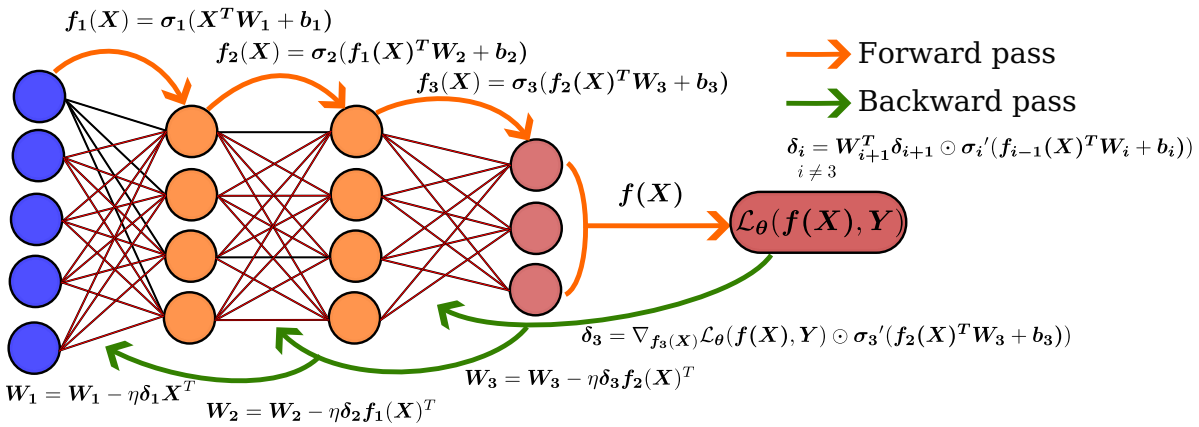


Figure 1.4 – Schematic representation of the forward and backward propagation in an MLP.

However, even with this method, computing the gradient over the entire training set at each iteration can become computationally expensive as the dataset size increases. One solution is to rely on Stochastic Gradient Descent (SGD), an extension of gradient descent, to estimate the gradient and update the parameters for each of the j data points such as

$$\boldsymbol{\theta_{i+1}} = \boldsymbol{\theta_i} - \eta \nabla_{\boldsymbol{\theta_i}} l(f_{\boldsymbol{\theta_i}}(\boldsymbol{X_j}), \boldsymbol{Y_j}). \tag{1.10}$$

While this method can be more efficient from a computational point of view, it leads to noisy estimations of the gradient, which deteriorates the performance of the algorithms. An alternative is to combine both approaches into what is referred to as Mini-Batch Gradient Descent (MBGD). In this scenario, the training dataset is split into *batches*, and optimization is performed iteratively on each batch.

$$\boldsymbol{\theta_{i+1}} = \boldsymbol{\theta_i} - \eta \nabla_{\boldsymbol{\theta_i}} \mathcal{L}_B(\boldsymbol{\theta_i}, \boldsymbol{X_B}, \boldsymbol{Y_B}) \tag{1.11}$$

with the function $\mathcal{L}_B(\boldsymbol{\theta_i}, \boldsymbol{X_B}, \boldsymbol{Y_B})$ corresponding to the loss function averaged over the whole batch B at iteration i, and $\boldsymbol{X_B}$ (resp. $\boldsymbol{Y_B}$) the batch of inputs (resp. labels).

This method combines the best aspects of both previous approaches by ensuring that the gradient is not estimated on a large sample of data, while allowing for a more accurate estimation of the gradient than those made with only one data point. In MBGD, the process of going over all batches is called an *epoch*. These methods allow for the search for a set of parameters that minimize the loss function Equation 1.5. However, contrary to the case with convex loss functions, there is no guarantee of convergence to the optimal set of parameters [GBC16]. In particular, a large number of factors, such as the choice of hyperparameters, the initialization of the parameters, *etc.*, impact the convergence of the algorithms, and more globally, the training process.

1.2.3 Optimisation Challenges

The most visible hyperparameters influencing the performance of the learning algorithm are those that directly impact Equation 1.8, such as the learning rate η, the number of epochs, or the initial value of $\boldsymbol{\theta}$. Their impact on the training process, as well as that of the parameters initialization, is represented in Figure 1.5. In this figure, gradient descent is used to find the minimum of a non-convex function, but we vary the configuration to highlight the importance of these hyperparameters.

This figure highlights several phenomena that can occur when optimizing a non-convex function. Firstly, we can observe that while gradient descent tries to optimize the value of the parameters to minimize the loss function, this process does not guarantee to find the optimal parameters that correspond to the minimum of the loss function. Indeed, as we can see in Figure 1.5, the gradient descent can either not converge at all (Figure 1.5b), or converge to a sub-optimal point, *i.e.*, a local minimum (Figure 1.5a).

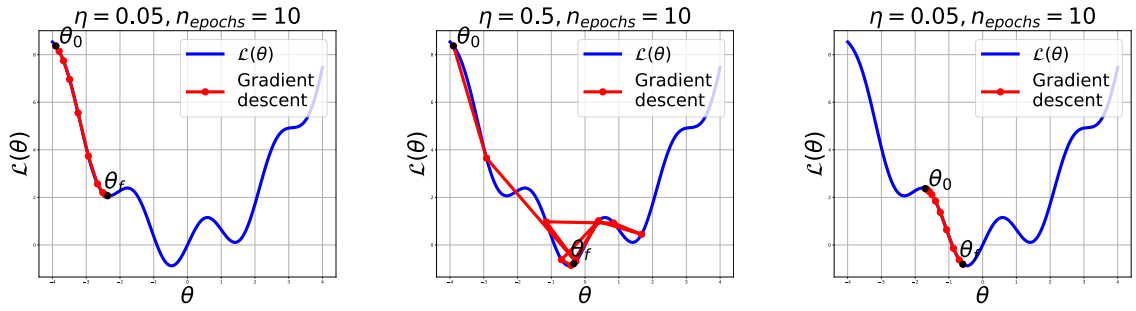
Definition 1.9 (Local minimum) *Let $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ be a parametric model, $D_{\boldsymbol{X},\boldsymbol{Y}}$ a dataset of inputs $\boldsymbol{X} \in \mathcal{X}$ and their corresponding labels $\boldsymbol{Y} \in \mathcal{Y}$, and $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y})$ the loss function associated with this dataset and this model. A local minimum is a point $\boldsymbol{\theta} \in \Theta$ for which there exist an $\epsilon > 0$ such that $\forall \boldsymbol{\theta'} \in \Theta$ respecting $||\boldsymbol{\theta'} - \boldsymbol{\theta}||_2 < \epsilon$ the following inequality holds:*

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) \leq \mathcal{L}(\boldsymbol{\theta'}, \boldsymbol{X}, \boldsymbol{Y}) \tag{1.12}$$

The apparition of such points is due to the non-convex nature of the loss function. Local minima are a threat during training because they correspond to stationary points, *i.e.*, points at which the gradient descent can stop, while potentially being far from the global minimum, which corresponds to the optimal solution.

Definition 1.10 (Global minimum) *Let $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ be a parametric model, $D_{\boldsymbol{X},\boldsymbol{Y}}$ a dataset of inputs $\boldsymbol{X} \in \mathcal{X}$ and their corresponding labels $\boldsymbol{Y} \in \mathcal{Y}$, and $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y})$ the loss function associated with this dataset and this model. A global minimum is a point $\boldsymbol{\theta} \in \Theta$ for which the following inequality holds:*

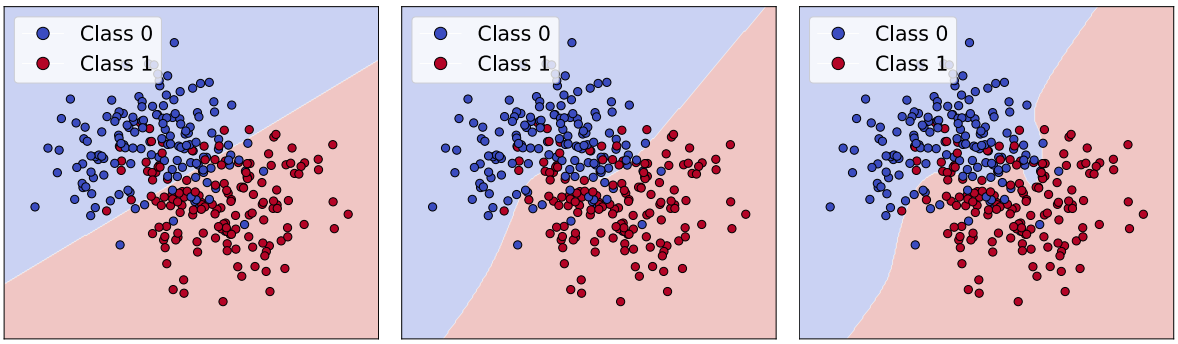
$$\forall \boldsymbol{\theta'} \in \Theta, \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) \leq \mathcal{L}(\boldsymbol{\theta'}, \boldsymbol{X}, \boldsymbol{Y}) \tag{1.13}$$



(a) Local minimum. (b) No convergence. (c) Global minimum.

Figure 1.5 – Finding the minimum of a non-convex function under different configurations with gradient descent.

In Figure 1.5c, the algorithm converges toward the global minimum of the function, which corresponds to the optimal parameters. The last phenomenon represented in Figure 1.5b corresponds to the case where the learning rate value is too large, leading to instability during the learning process. All these situations highlight the fact that the hyperparameter search is a critical step in the training phase.



(a) Underfitting (degree 1). (b) Good fit (degree 3). (c) Overfitting (degree 15).

Figure 1.6 – Comparison between underfitting, overfitting and a good fit for a polynomial model.

However, there are other hyperparameters, less direct than the learning rate or the number of epochs, that influence the learning process. In particular, as we mentioned in Subsection 1.1.1, the perceptron is fundamentally limited by its architecture, and the same limitation applies to MLPs. This limitation is connected to the *capacity* of the model, which describes its ability to fit a large set of functions. If the capacity of the model is too low, then the algorithm will be unable to represent the underlying distribution in the training data and have poor performance, this is *underfitting*. On the other hand, if the model capacity is too large for the given task, then the approximated function will overfit on the training data, leading to poor generalization performance, this is known as *overfitting*. In Figure 1.6, we plot the decision boundary learned by three polynomial models, each of a different degree, on a binary classification task.

We can see the difference between underfitting and overfitting compared to a good fit. In Figure 1.6a, we use a polynomial model of degree 1, resulting in a linear separation between the two classes. However, we can clearly see from the data distribution that this is not optimal, as many points are misclassified. The model capacity is too low for this task, and the model underfits. One solution is to increase the degree of the polynomial model so that it can learn more complex functions, creating a better separation between the two classes. This alternative is illustrated in Figure 1.6c, where the degree of the model is now set to 15. While the number of misclassifications is lower compared to the model in Figure 1.6a, we can observe that the decision boundary learned by the model is highly impacted by the training data with some irregularities that will decrease the generalization performance of the algorithm on new data. The model capacity is too

high for the task, and the model overfits on the training data. Finally, in Figure 1.6b, we reduced the degree of the polynomial model to 3, which allows the model to learn a sufficiently complex function so that the performance is good on the training set while limiting the risk of overfitting on the training data.

Overfitting and underfitting correspond to the two main challenges in ML and, therefore, in DL. However, it is possible to detect them during training by tracking the losses on the training and testing datasets. This is achieved by evaluating the loss function at each epoch and tracking its evolution during the training process. Indeed, in case of underfitting, the model will not be able to approximate correctly a function that will discriminate the classes, and the training loss will plateau. And once the model overfits on the training data, it loses in generalizability. Therefore, it is standard practice to combine the training loss with a loss evaluated on unseen data, *i.e.*, data that are not included in the training set and on which the model has not been trained. That way, if the loss function estimated on the training data decreases while the one computed on the unseen data plateaus or even increases, it is likely that the algorithm overfits on the training data. In a complementary way to the training dataset, the unseen data are referred to as the test data, and the loss evaluated on these corresponds to the test loss. To create both datasets, it is common to split a single global dataset into training and testing datasets. That way, both of these come from the same distribution.

Once this issue is detected, several methods can be employed to mitigate it. As we previously mentioned, changing the capacity of the model can help prevent both overfitting and underfitting. However, in the case of overfitting, this is not always optimal or possible, depending on the task. Therefore, alternative methods to prevent this must be found. One of the most useful is *regularisation*. Informally, regularisation consists of penalizing certain types of functions based on some previous hypothesis. For example, one common hypothesis is that solutions with large parameters are less optimal than solutions with close to zero parameters. More practically, this is done by adding a constraint in the loss function. Therefore Equation 1.5 becomes

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{N} \sum_{j=1}^{N} \texttt{loss}(f_{\boldsymbol{\theta}}(\boldsymbol{X}_j), \boldsymbol{Y}_j) + \lambda \cdot \texttt{reg}(f_{\boldsymbol{\theta}}) \tag{1.14}$$

with $\texttt{reg}(f_{\boldsymbol{\theta}})$ a penalty function, or regularizer, and $\lambda \in \mathbb{R}^+$ an hyperparameter controlling the influence of this regularisation. In practice, $\texttt{reg}(f_{\boldsymbol{\theta}})$ is usually the p-norm $||.||_p$ on the parameters of the model $f_{\boldsymbol{\theta}}$. Both L_1 and L_2 norms have been widely used as regularizers

and early models, such as the Least Absolute Shrinkage and Selection Operator (LASSO) [Tib11], and Ridge regression [HK70], correspond to famous use cases. Adding a penalty to the loss function corresponds to an explicit regularization. However, it is possible to have implicit regularization. One of the most popular methods for implicit regularization is Batch Normalization (BN) [IS15], which normalizes each batch B during MBGD such as

$$\texttt{BN}(\boldsymbol{X}) = \gamma \cdot \frac{\boldsymbol{X} - \mu_B}{\sigma_B} + \beta \tag{1.15}$$

with γ and β both real parameters learned during training, and μ_B (resp. σ_B) the mean (resp. the standard deviation) of the batch B. While these methods correspond to some of the main explicit and implicit regularization methods, several other methods, such as the early stopping [Pre12], introducing dropout layers [Sri+14], or even data augmentation, can be successfully applied during training.

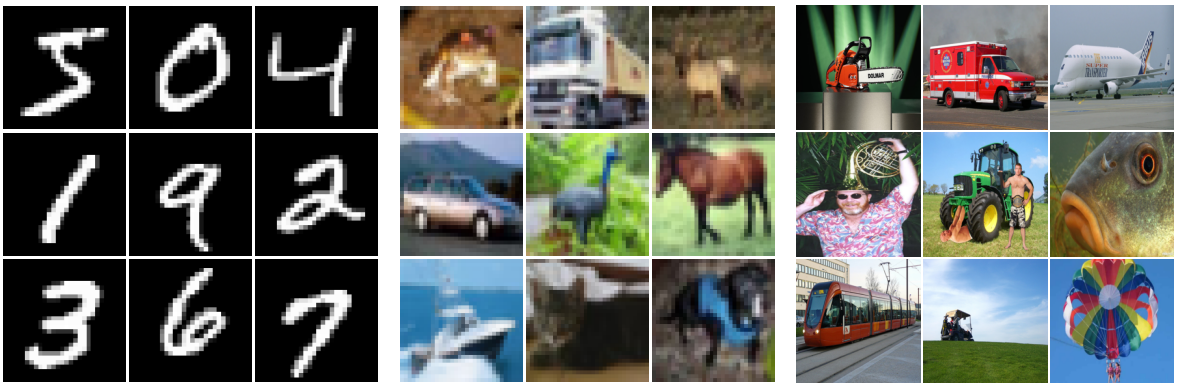
1.2.4 Performance Evaluation and Common Datasets

After the training, the global performance of the algorithm is evaluated on the task for which it was designed. For regression tasks, it is common to use the MSE both as a loss function during training and as an evaluation function after training. For classification tasks, it is somewhat different. The training is performed on a function that is easily differentiable. However, such functions are not the most suitable for evaluating the performance on the task. Especially for image classification, more tangible metrics can be used, and the most common one is *accuracy*:

Definition 1.11 (Accuracy) *Accuracy correspond to the percentage of agreement between the predicted class by the model and the true label.*

Several alternative metrics can be derived from accuracy, such as the confusion matrix or *top-k* accuracy. However, since in this thesis we will mainly use accuracy, we do not provide further details on these metrics.

With these improvements in architecture and training methods, the field of ML was able to tackle more complex tasks. This is especially visible for classification tasks, which have evolved from classifying the IRIS dataset [Fis36], with three classes and only two features, to the MNIST [Den12] and CIFAR-10 [KH09] datasets, featurings 28×28 grayscale and colored images and 10 classes, and ultimately to the ImageNet dataset [Den+09]



(a) Images from MNIST. (b) Images from CIFAR-10. (c) Images from ImageNet.

Figure 1.7 – Subsets of the MNIST, CIFAR-10 and ImageNet datsets.

with its 1000 classes. Figure 1.7 illustrates extracts of the main datasets used in this thesis (MNIST, CIFAR, and ImageNet).

However, even with these improvements, these tasks remain difficult for MLPs. Even for the CIFAR-10 dataset, which is considerably simpler than the ImageNet dataset, an MLP with 4 hidden layers and nearly a million parameters and basic training will only achieve an accuracy of around 50%. These results highlight the limitations of the MLP and justifies the need for more complex architectures based on different layers such as the Convolutional Neural Network (CNN) which will be presented in the next chapter.

Chapter 2

Deep Neural Networks in Embedded Systems

Now that we have presented the basic building blocks of the perceptron and how they led to the MLP, we can move on to more modern architectures designed for more complex tasks. In this chapter, we will first introduce the basic layers of DNNs as well as some of the most commonly used architectures. We will then describe the life cycle of DNNs from data collection to deployment. Finally, we will focus on the constraints introduced by this environment on DNNs and how both the hardware and the design of DNNs have adapted to these.

Contents

2.1 Deep Neural Networks 25

2.1.1 Parametric Layers 25

2.1.2 Non-Parametric Layers 28

2.1.3 Complex Architectures 29

2.1.4 Life Cycle of Deep Neural Networks 30

2.2 Deep Neural Networks on Embedded Systems 31

2.2.1 Hardware Constraints 31

2.2.2 Layer Adaptations 32

2.2.3 Quantization 34

2.2.4 Other Compression Methods 37

2.2.5 Hardware Architectures for Embedded Deep Neural Networks . 38

2.1 Deep Neural Networks

2.1.1 Parametric Layers

So far in this thesis, we have only considered fully connected layers in MLPs. However, several other types of layers have been introduced over the years. In this section, we specifically focus on *parametric* layers, *i.e.*, layers containing parameters that are updated during the learning algorithm. One of the most popular parametric layers is the convolutional layer, which forms the main block of CNNs and is widely popular on tasks including images or any grid-like data. Such layers were inspired by the visual system of animals and rely on three fundamental properties: locality, shared weights, and temporal or spatial subsampling [Lec+98].

Convolutional layers apply a *filter*, or *kernel*, on an *input feature map* to produce an *output feature map*. The application of the filter corresponds to the sum of the element-wise product between the filter and the value of the input feature map. In a convolution, the filter slides horizontally and vertically over the entire input feature map, typically from the top-left corner to the bottom-right corner. This operation is more suitable than the classical matrix product on grid data, as it enables the detection of local patterns in the input, such as an edge, for example. Furthermore, by sliding over the entire input feature map, all similar patterns can be detected by a single filter independently of their localization in the input. While convolution can be generalized to higher dimensions, most machine learning algorithms use two or three-dimensional convolution. We provide a schematic representation of a two-dimensional convolution of 4×4 input by a 3×3 kernel in Figure 2.1.

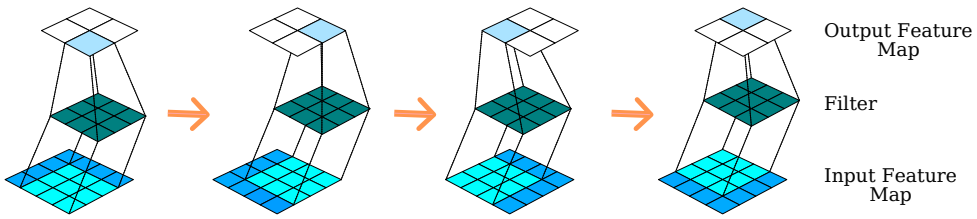


Figure 2.1 – Schematic representation of a two-dimensional convolution.

In the example represented in Figure 2.1, the convolution is performed on an input corresponding to a 2D grid. This is equivalent to a black and gray image, where the two dimensions correspond to the height H_l and the width W_l of the image. However, for color

images, a third dimension appears corresponding to the encoding of the color, which is usually achieved using three channels. Since it is frequent for convolution to be applied to a tensor with more than one channel, similar to colored images, the processing is adapted to take into account an arbitrary number of channels C_l. More formally, we can define a convolutional layer f_l in the following manner:

Definition 2.1 (Convolutional Layer) *Let $\boldsymbol{X} \in \mathbb{R}^{C_l \times H_l \times W_l}$ be the input feature map of layer l, $\boldsymbol{Y} \in \mathbb{R}^{H_{l+1} \times W_{l+1}}$ the output feature map of layer l, $\boldsymbol{W} \in \mathbb{R}^{C_l \times K_h \times K_w}$ a weights tensor composed of C_l channels each with a height (resp. width) of K_h (resp. K_w) and $\sigma : \mathbb{R} \to \mathbb{R}$ a non-linear activation function applied element-wise. The convolutional layer l corresponds to the following function $f_l : \mathbb{R}^{C_l \times H_l \times W_l} \to \mathbb{R}^{H_{l+1} \times W_{l+1}}$ such that $\forall (x, y) \in (H_{l+1} \times W_{l+1})$ the following equation is respected:*

$$\boldsymbol{Y}_{x,y} = \sigma(\boldsymbol{W} * \boldsymbol{X}_{x,y}) = \sigma\Big(\sum_{c=0}^{C_l-1} \sum_{i=0}^{K_h-1} \sum_{j=0}^{K_w-1} \boldsymbol{W}_{c,i,j} \cdot \boldsymbol{X}_{c,x+i,y+j}\Big), \tag{2.1}$$

with $H_{l+1} = H_l - K_h + 1$, $W_{l+1} = W_l - K_w + 1$.

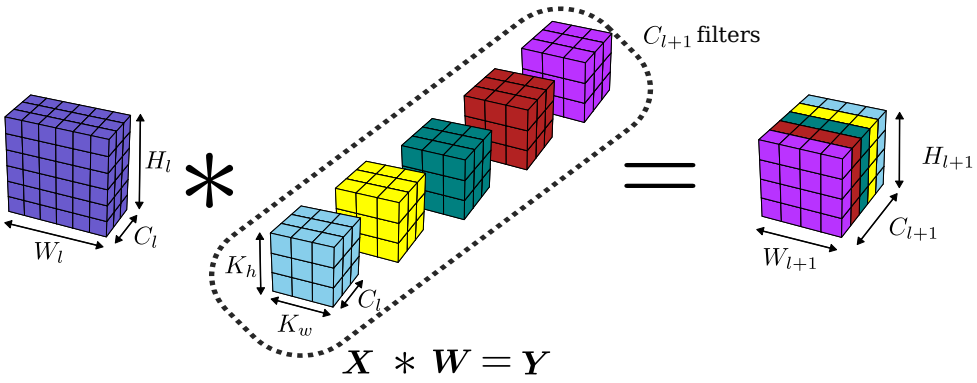


Figure 2.2 – Schematic representation of an input feature map composed of $C_l = 3$ channels with $H_l = 6$ and $W_l = 6$ on which is applied a convolutional layer with 5 filters for which $K_h = 3$ and $K_w = 3$.

Similarly to fully connected layers, convolutional layers correspond to a linear transformation, and therefore, they are also followed by a non-linear activation function. Definition 2.1 introduces the definition of a convolutional layer with only one filter or kernel. However, the number of kernels can be arbitrarily set to $C_o \in \mathbb{N}$ during the definition of the model's architecture. In this case, the weights tensor describing the convolutional

layer f_l is now usually described with a fourth dimension corresponding to the number of output channels, *i.e.*, $\boldsymbol{W} \in \mathbb{R}^{C_o \times C_l \times K_h \times K_w}$. A schematic description of a convolutional layer with four filters is provided in Figure 2.2.

One consequence of convolutional layers is the reduction in size of the input feature map. Indeed, since we are performing elementwise multiplication, the features close to the border in the input feature map might not have been included in the convolution, resulting in a smaller output feature map. To reduce this impact, it is common to use *zero-padding* around the input feature map, as represented in Figure 2.3.

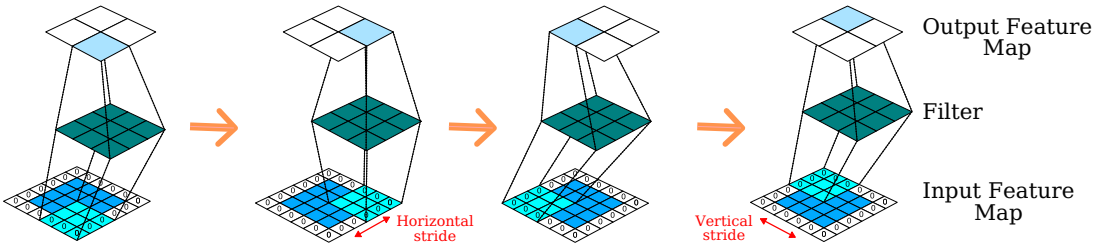


Figure 2.3 – Schematic representation of a convolution with zero-padding and a stride of 3.

Figure 2.3 illustrates another hyperparameter in convolutional layers: the *stride* that controls the steps between each application of the filters. Both the srtide and the padding impact the dimension of the output feature map through the following equations:

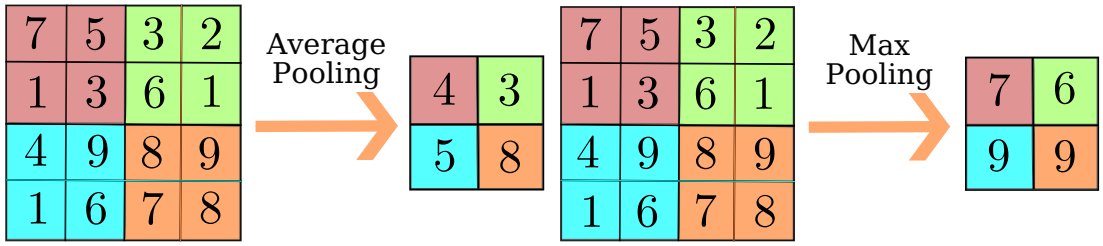
$$H_{l+1} = \left\lfloor \frac{H_l - K_h + 2P_h}{S_h} \right\rfloor + 1, W_{l+1} = \left\lfloor \frac{W_l - K_w + 2P_w}{S_w} \right\rfloor + 1, \tag{2.2}$$

using the same notations as in Definition 2.1, with P_h (resp. P_w) $\in \mathbb{N}$ the horizontal (resp. vertical) padding, and S_h (resp. S_w) $\in \mathbb{N}$ the horizontal (resp. vertical) stride. In theory, the padding and the stride can be different horizontally and vertically, but in practice, they are often the same, and we only use S and P when there is no confusion.

While convolutional layers are still one of the most used types of layers in DNNs today, the search for new types of parametric layers remains an active research field, which has led to the development of recurrent layers [HS97] and transformer layers, [Vas+17], for example. Due to their trainable parameters, parametric layers correspond to the core of DNNs. However, several *non-parametric* layers have also been introduced, which greatly improve the performance of DNNs.

2.1.2 Non-Parametric Layers

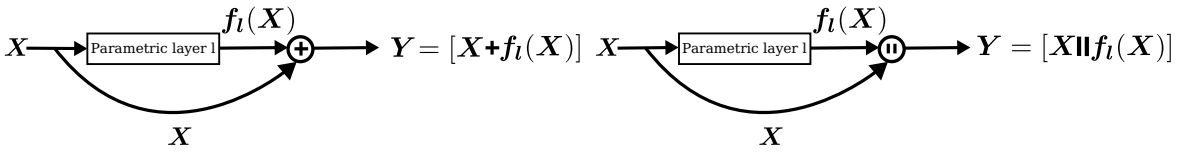
Several essential elements of modern architectures involving convolutional layers are non-parametric layers. In this subsection, we will introduce two of those, namely *pooling layers* and *residual connections*. Pooling layers reduce the dimension of the feature maps by further aggregating features in a specific area. These layers are intrinsically connected to convolutional layers as they were introduced complementarily in [Lec+98] and are responsible for the third fundamental property of CNNs presented in Subsection 2.1.1: subsampling. Combining pooling layers and convolutional layers sequentially enables us to move from low-level representations to high-dimensional features. This aggregation can be performed in different ways. In this thesis, we will only consider two, which correspond to the most commonly used operations in modern architectures [SZ14; He+15b; How+17], averaging and taking the maximum within a given volume. These types of pooling are usually referred to as *AveragePooling* and *MaxPooling* layers, respectively. In Figure 2.4, we illustrate the application of two pooling layers, an AveragePooling (see Figure 2.4a) and a MaxPooling (see Figure 2.4b), on a four-by-four input feature map.



(a) 2×2 Average pooling. (b) 2×2 Maximum pooling.

Figure 2.4 – Schematic representation of the application of a 2×2 pooling layer on a 4×4 input feature map.

Non-parametric layers can also be designed to improve the learning algorithm. This is the case for *residual blocks* [He+15b]. These blocks introduce a residual connection between two layers to help the training of very deep architectures. We represent in Figure 2.5a a residual connection between two layers in a model composed of fully connected layers. In [He+15b], the residual connection corresponds to a summation between the current and a previous layer. Therefore, no parameters are included in this layer. However, residual connections can be extended to include parameters or even change the operation by combining the two layers. This is done in [HLW16], where the idea is still to ease the



(a) Residual connection [He+15b]. (b) Dense connection.

Figure 2.5 – Schematic description of a residual and a dense connection.

propagation of the gradient, but the residual connections are no longer summed with the current layer, but concatenated as illustrated in Figure 2.5b, these are referred to as *dense blocks*.

Similarly to convolutional and pooling layers, residual or dense blocks are complementary to convolutional or fully connected layers. More generally, both parametric and non-parametric layers are combined in modern architectures, and this arrangement or architecture defines the model. Over the years, several architectures based on a similar backbone have demonstrated impressive results on benchmark tasks such as the ImageNet dataset. In the next subsection, we will present more complex architectures and introduce some of these families of DNNs.

2.1.3 Complex Architectures

In this thesis, we will solely focus on CNNs and MLPs. As previously mentioned, several subfamilies of CNNs exist, but to understand the different classifications between these, it is first necessary to properly introduce the general architecture of a CNN. Contrary to MLPs, which only use one type of layer,*i.e.*, fully connected layers, CNNs use different kinds of layers and can usually be split into two parts, as represented in Figure 2.6. A first block involving a series of layers alternating between one or several convolutional layers and a pooling layer, and a second block composed of fully connected layers. This design is especially useful for image classification, where the first block selects the discriminative features in the input images and the second block uses these features to classify the images.

Figure 2.6 illustrates the general architecture of a CNN. However, with the development of new layers, such as the residual connection, some modifications to the main architecture have been introduced. These modifications were first introduced in a specific model, but due to the popularity and the performance of the original model, several variants, all sharing the same backbone, were successively introduced, leading to the creation

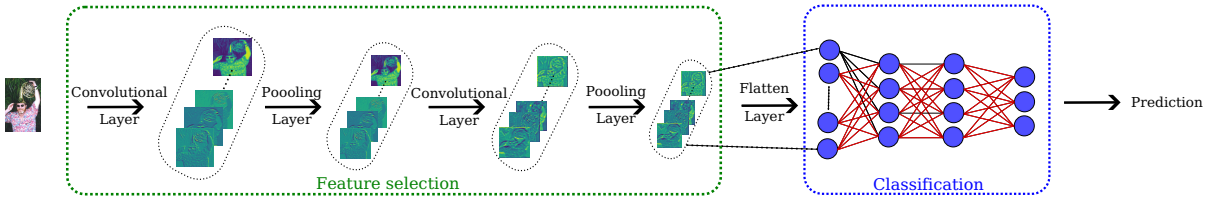


Figure 2.6 – Convolutional Neural Netwok performing images classifications on ImageNet.

of subfamilies of CNNs. In this thesis, we will focus on three of those, more specifically, Very Deep Convolutional Networks, also known as VGG models [SZ14], ResNet [He+15b] models, and MobileNet [How+17] models. The VGG subfamily is a group of models with varying depths based on 3×3 convolution filters and max pooling layers. ResNets are a subtype of CNNs that introduced the residual blocks to improve the training of very deep architectures. This subfamily also includes networks of different depths, ranging from 18 to 152 layers. Finally, MobileNets are CNNs specifically designed to be more efficient in terms of computations. Further details on this subfamily will be provided in Subsection 2.2.2.

All these architectures were designed to improve SOTA results on the ImageNet contest. As a consequence, achieving similar performance to the one claimed in the original paper by training them from scratch, is a time-consuming and challenging process. Therefore, it has become common for complex architecture significantly improving the previous performance on a generic task, *e.g.*, the ImageNet contest, to become publicly accessible along with their trained parameters as downloadable pre-trained architectures. This public access to pre-trained networks has significantly impacted the life cycle of DNNs.

2.1.4 Life Cycle of Deep Neural Networks

Traditionally, the life cycle of DNNs in AI begins with the definition of a task and the collection of data relevant to that task. Then, this step is usually followed by a form of preprocessing on the data, which can include data augmentation, labeling the data, *etc.* Once the dataset has been constructed, the design of the model can start. This step is usually experimental and iterative. This process alternates between a phase of architecture selection and training, with a phase of fine-tuning and improvements of the architecture based on the performance of the algorithm on the test set.

However, as previously mentioned, the difficulty of the tasks has led to the need for deeper architecture, making it very difficult for an individual to train a model from scratch. One consequence of this, which we mentioned in Subsection 2.1.2, is the creation of families of architecture with benchmark models trained by large companies with specialized resources. The models introduced in Subsection 2.1.3 correspond to examples of such benchmark models and are directly downloadable and ready to use by a user without having to perform the full training process. But another use case is to start from these pre-trained models and fine-tune them for a specific targeted task. The idea is that the complex representation learned by the pre-trained models can be generalized to a different task [Yos+14]. For example, in image classification, the detection of some low-level features, such as edges or certain forms, is likely to be still useful even if the dataset of the final task is different.

Once the model achieves sufficient performance on the targeted task, it must be deployed to make it accessible to users. We can broadly classify the deployment of DNNs into two classes: the first one is deployment on cloud-based solutions, such as in Machine-Learning-as-a-Service (MLaaS) [RGC15], and the second one is deployment at the edge on embedded devices. Depending on the type of constraint on the application, including the model, both approaches can be considered, and a large number of examples of each solution have emerged in the past decade. However, in the context of this thesis, we will focus on DNNs deployed on embedded devices. As the deployment of DNNs on embedded devices introduces new challenges, we will describe in the following section the reasons and adaptations to these challenges for this deployment.

2.2 Deep Neural Networks on Embedded Systems

2.2.1 Hardware Constraints

While MLaaS has gained popularity through the introduction of large language models, cloud deployment for DNNs is not always possible. Indeed, when the model is used in an application with critical time constraints, sending the data to a cloud-based model is not an option. Furthermore, sending data to a third party might not always be possible, especially in the context of confidential data. Therefore, edge AI becomes necessary as soon as there is a need for efficient or private inference.

However, embedded systems impose several additional constraints on DNNs. The two

main constraints are *memory size* and *energy consumption* [Sze+17]:

- Memory size: As previously mentioned, modern DNNs use an ever-larger number of parameters, requiring an increasing memory size to store them. Furthermore, DNNs rely on the computation of numerous intermediate values, which also require storage, highlighting the critical importance of memory for DNNs .
- Energy consumption: the large number of weights associated with the sequential nature of DNNs implies that a straightforward implementation of DNNs will require a lot of memory access and computations, which is very costly in terms of energy.

Due to these reasons, several methods have emerged in the past decade to deploy DNNs on embedded devices without compromising the model's accuracy on the task. Conjointly specialized hardware accelerators dedicated to DNNs were introduced at the same time. In the following sections, we will introduce some of these adaptations.

2.2.2 Layer Adaptations

To improve the deployment of DNNs on embedded devices, one straightforward method is to modify the computation directly in the layers to reduce the number of operations or parameters. One of the most famous examples of such adaptation is the substitution of traditional convolutions with *depthwise separable convolutions*. This alternative was first introduced in [How+17], but it proved so effective that it led to the creation of the MobileNet subfamily [San+18; How+19; Qin+24]. The idea is to split the convolution layer into two layers, one performing *depthwise* convolution and another performing *pointwise* convolution.

Depthwise convolutions correspond to a subtype of *grouped* convolutions. Grouped convolutions differ from classical convolutions in the way they handle depth. In classical convolutions, the number of channels in the kernels is implicitly set to the number of channels in the input feature map. That way, when a filter is convoluted over the input feature map, the output feature map is a two-dimensional tensor. In a grouped convolution, this is no longer the case, and convolutions can be grouped by channel so that a given filter can only be applied to specific channels of the input feature map. Depthwise convolution is a special case of grouped convolution, where the number of groups is equal to the number of channels. Therefore, each filter in a depthwise convolution is two-dimensional, and the number of filters is equal to the depth of the input feature map. Similar to classical convolutions, the shape of the output feature map of depthwise convolution can be modulated by the padding and stride hyperparameters.

In depthwise separable convolution, this operation is then followed by a pointwise convolution. In pointwise convolutions, the output feature maps from the depthwise convolutions are concatenated into a three-dimensional tensor (four in the case of batched inputs) and then processed by a $1 \times 1 \times c$ filter, where c is the number of groups in the depthwise convolution. Each filter is applied over the complete three-dimensional tensor, producing a two-dimensional tensor whose shape depends on the stride. Since this step performs $1 \times 1 \times c$ convolution, the padding parameter is irrelevant. Finally, since each filter produces a tensor, the number of output channels of the depthwise separable convolution corresponds to the number of pointwise filters.

Although composed of two distinct operations, the composition of depthwise and pointwise convolution is often seen as a single layer in most descriptions. A schematic description of the complete process of a depthwise separable convolution is presented in Figure 2.7 on a $3 \times 5 \times 5$ with only three filters in the pointwise convolution. Since there is no padding in the depthwise convolution and the stride of the pointwise convolution is set to one, the output shape is $3 \times 3 \times 3$.

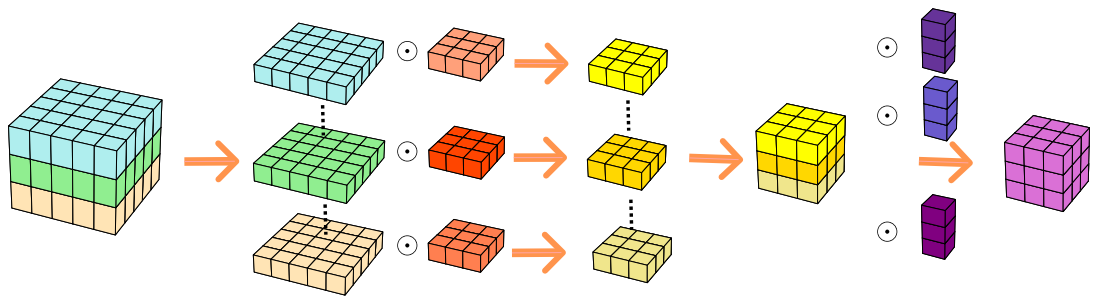


Figure 2.7 – Schematic description of the depthwise separable convolution.

To understand the benefits of depthwise separable convolution, it is helpful to quantify the number of parameters needed to produce the same output feature maps with a classical convolution in the scenario presented in Figure 2.7. Using classical convolutions, 81 parameters would have been necessary (three kernels of shape $3 \times 3 \times 3$). While with depthwise separable convolutions, only 36 parameters are needed, three kernels of shape 3×3, and only 9 parameters for the pointwise convolutions due to the 3 $3 \times 1 \times 1$-kernels. Therefore, the total number of parameters required for this operation is divided by 2.25, which reduces both the size of the model and the number of intermediate computations.

This reduction in the number of parameters in the depthwise separable convolution has increased the possibility of deep architectures on embedded systems and, therefore,

their performance on complex datasets, such as the ImageNet dataset. For these reasons, several other variations of MobileNets have emerged, *e.g.*, MobileNetv2, MobileNetv3, and this subfamily remains one of the most popular in embedded systems [Zha+22]. While this adaptation of the operation is one of the most famous, such modification is not possible on all layers. Therefore, several generic methods to reduce the hardware cost of DNNs have been introduced in the past few years. In the next section, we describe the most famous one: *quantization*.

2.2.3 Quantization

Quantization is a process that consists of reducing the number of bits associated with the representation of the processed value. It originates from the need to represent real values in physical systems using only a finite number of bits. Quantization was successfully applied to DNNs to reduce their size and computational complexity. Indeed, traditionally, DNN inference was performed using 32-bit data floating point values following the IEEE-754 norm, *i.e.*, a variable x in simple precision is encoded using a bit for the sign S_a, 8 exponent bits E_a, and 23 bits for the mantissa M_a such that:

$$x = (-1)^{S_x} \times 2^{E_x - 127} \times (1 + 2^{-23} \times M_x). \tag{2.3}$$

with E_x and M_x represented as unsigned integers in a binary format. However, this representation enables the encoding of values with a large dynamic range from 10^{-38} to 10^{+38}, which is not optimal for DNNs, as the distribution of the parameters tends to form a Gaussian centered around zero due to the training algorithm and the normal initialization of the parameters [He+15a]. Therefore, one straightforward solution would be to represent only the value in an interval $[-N, N]$ with $N \in \mathbb{R}$ and $N \ll 10^{+38}$. This would discard all values not included in the range $[-N, N]$, but as mentioned, such values are unlikely if N is sufficiently large. Since this interval is considerably smaller than the full range of the 32-bit floating point representation, it is possible to reduce the number of bits used to represent it. Furthermore, it is possible to reduce the number of bits even further as DNNs have demonstrated a good tolerance to errors [Gup+15].

One way to exploit this property is to subdivide the interval $[-N, N]$ into subparts of the same size and represent only one value by subpart. All the used values, *e.g.*, are then rounded to the nearest value. We illustrate this process in Figure 2.8.

By discarding these values, we introduce a small error and different weights can be

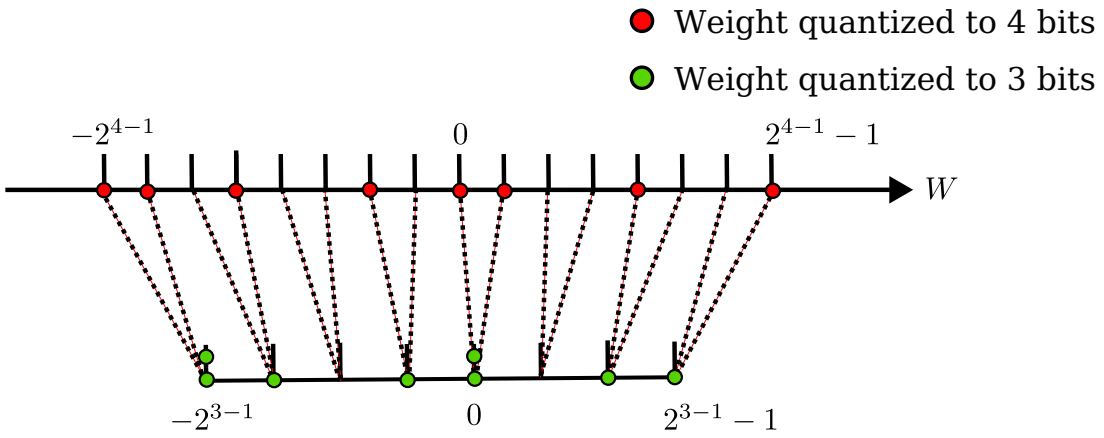


Figure 2.8 – Schematic description of the quantization of the eight weights of neuron from a format using 4 bits to 3 bits.

rounded to the same value as shown in Figure 2.8, but this allows the developer to use an even lower representation, as the size of subparts depends on the number of bits used to represent the internal value of the DNN. This process corresponds to a toy example of quantization. However, it highlights several critical points in quantization.

First, contrary to depthwise separable convolutions, which are layer adaptations and, consequently, cannot be applied to all architectures, quantization is more generic and is not restricted to a specific architecture, such as CNNs. Secondly, due to its impact on the manipulated values, such as the weights or the inputs, quantization can deteriorate the accuracy of the model on the task for which it was designed. And finally, from the type of rounding to the choice of variable to consider, in quantization, there are numerous methods and hyperparameters to consider. In this thesis, we will focus on two types of these: first, the *quantization target*, and then the choice of the global *quantization method*.

In our toy example, we have only discussed the quantization of the model's weights. However, quantization can be applied to any data within the DNN, such as the parameters or the activation values. Still, it can also be extended to the input of the DNN. Furthermore, to mitigate the impact of quantization on the accuracy of the model, it may be necessary to adjust the quantization level for each layer [Blo+18; Che+18]. Indeed, as mentioned in Subsection 2.1.1, in convolutional layers, the first layers are usually associated with low-level representation features directly present in the inputs, while high-level features often correspond to deeper layers. This distinction can be exploited to adapt the

number of bits allocated to each layer individually [LTA18]. Furthermore, in our toy example, we have also left out the discussion as to when quantization is applied. The most straightforward method is to train the model using full-precision data and then apply quantization to the trained weights. This method is very similar to the process that we illustrated in Figure 2.8. One downside of this method is that it does not account for the final restriction on the range of possible values for the parameters, and therefore, it can lead to a significant drop in accuracy. To mitigate this issue, it is common to include a final step where the model is fine-tuned on a subset of the data for a small number of epochs. This approach corresponds to Post Training Quantization (PTQ) [BNS19; Mel+19]. On the other hand, an alternative is to consider quantization directly during the training stage, which is Quantization Aware Training (QAT) [CBD15]. In QAT, so as not to deteriorate excessively the performance of the training algorithm, it is frequent to quantize only the forward pass and keep full precision on the backward pass. The non-differentiable operations induced by the quantization process are usually modeled by straight-through estimators [BLC13], and as such, the training process can be performed similarly to what has been introduced in Subsection 1.2.2. QAT is typically more costly as it necessitates retraining the model from scratch, but it usually improves the accuracy with lower representation compared to PTQ [Nag+21].

Quantization directly improves the memory footprint of DNNs by reducing the number of bits required to represent the weights, but also the performance during inference. This co-benefit is first due to the lower number of data transfers, but also to the possibility of using less energy-consuming data representation, such as fixed-point arithmetic [Hor14]. For example, using only variables represented with power of 2s, it is possible to implement multiplications as bit shifts. The extreme version of this has been achieved with Binarized Neural Network (BNN) [CBD15], which correspond to DNNs quantized to only one bit. In this scenario, weight multiplications are equivalent to addition, which significantly improves performance. Without going this far in the quantization, several results have been demonstrated with 16-bit and 8-bit quantization, and the design of efficient representations for quantization of DNNs is therefore a very active field.

Quantization directly compresses DNNs by reducing the representation of the used data. As such, it can be combined with other types of compression methods, such as depthwise separable convolutions. However, these two methods are not the only ones, and

in the next section, we present other notable compression methods.

2.2.4 Other Compression Methods

To further compress DNNs, several other methods exist, but one of the most common is to adapt the model's architecture. Contrary to depthwise separable convolution, this is not achieved by changing the type of operation performed, but by directly modifying the model architecture. Indeed, wider and deeper models naturally have a larger memory footprint as well as more intermediate values to compute. Therefore, reducing the model size is a straightforward method to reduce the hardware cost. This operation can be included in the training phase or conducted after.

An example of this during the training phase is Neural Architecture Search (NAS) [ZL17; Che+18]. NAS corresponds to the process of automatically searching for an architecture to improve the performance on a given task while removing the difficult step of architecture design. NAS achieves this result by exploring different combinations of hyperparameters defined in a search space. However, NAS was successfully adapted to include hardware constraints, enabling automatic search for efficient architectures with a low hardware cost. One drawback of this method is that the automatic process can become computationally expensive as the size of the search space increases.

An alternative approach is to start from a large, trained model and apply compression methods, such as *knowledge distillation* [HVD15] or *pruning* [HMD15]. Knowledge distillation corresponds to the process of training a smaller model using the predictions of a larger one as labels. This method is based on the intuition that the soft-labels of the larger model encode enough information to enable the transfer of knowledge to the smaller model. Pruning, on the other hand, does not require retraining a new model from scratch but rather directly modifies the original larger model. It was introduced to improve the generalization of models [LDS89], but in modern applications it is used as a compression method. After the training phase, a model is pruned, *i.e.*, some of its parameters or even neurons are set to zero or removed, effectively reducing its size. To ensure that the performance of the pruned model does not drop significantly, it is common to include a final phase of retraining for a low number of epochs.

All these methods are effective in reducing the hardware cost of DNNs, but they correspond to software adaptations. In a complementary way, specialized hardware dedicated

to DL has emerged in the past few years to improve both the performance of DNNs and the possibility of embedding larger models.

2.2.5 Hardware Architectures for Embedded Deep Neural Networks

Since AlexNet [KSH12], the training of DNNs has been traditionally performed on high-performance servers with GPUs. However, as previously mentioned, the constraints of embedded systems have led to the emergence of specialized hardware architectures, as well as frameworks dedicated to deploying DNNs on these platforms. This includes GPUs, Tensor Processing Units (TPUs), and Field-Programmable Gate Arrays (FPGAs). GPUs are designed for parallel computing and have found a natural application with DNNs because most computations are associated with matrix products, which are highly parallelizable. However, their high power consumption makes them less appropriate for DNNs deployed on embedded devices compared to TPUs and FPGAs [BZR18]. Similar to GPUs, TPUs are specifically designed to accelerate matrix products through a systolic array. TPUs are Application-Specific Integrated Circuits (ASIC) introduced by Google to accelerate AI inference and training in Cloud environments, as well as edge AI with so-called edge TPUs. One downside of TPUs is that their nature constrains them to one specific task under a specific configuration. In particular, TPUs are designed to accelerate 8-bit computations and, as such, are not optimized for other arithmetics. On the other hand, FPGAs are reconfigurable integrated circuits and, therefore, are more modular than TPUs. This is a significant advantage when considering different levels of quantization for various layers of the DNNs. Consequently, FPGAs are increasingly used for embedded AI, and several frameworks have been designed to facilitate the deployment of DNNs on FPGAs. Such frameworks include FINN [Umu+16; Blo+18] or Vitis AI [1] from AMD-Xilinx.

However, in this thesis, we will mostly consider a different kind of embedding device for DNNs: MCUs. MCUs are already a popular choice in embedded devices due to their low cost and ease of programmability. However, with the interest of researchers aiming to develop specialized software kernels to accelerate DNN operations [LSC18] and the integration of specific accelerators with MCUs [2], MCUs are now becoming a popular choice for embedded AI. Furthermore, similarly to FPGAs, numerous frameworks have emerged

1. https://xilinx.github.io/Vitis-AI/3.5/html/index.html
2. https://www.st.com/en/microcontrollers-microprocessors/stm32n6-series.html

in the past few years to facilitate the deployment of DNNs on MCUs. In this thesis, we will more specifically consider two of them: one open-source, Neural Network on Microcontroller (NNoM) [Ma20], and one proprietary X-Cube-AI [3].

In this chapter, we presented the process of architecture design for modern DNNs and highlighted the difficulties of this process. The complexity, as well as the numerous adaptations required to deploy DNNs at the edge, justify the value of the IP associated with modern DNNs. Therefore, it is crucial to provide assurances regarding the security of deployed DNNs. However, the deployment of DNNs on embedded devices increases the attack surface available to an attacker, and developers must consider attacks specifically created for those devices: physical attacks.

3. https://www.st.com/en/embedded-software/x-cube-ai.html

Chapter 3

Side-Channel Analysis

The deployment of DNNs on embedded devices opened up the door to a specific kind of attack: physical attacks. Physical attacks were first introduced to target secure cryptosystems by exploiting the fact that the algorithm runs on a physical device and, as such, produces side-channel observables. However, they are not limited to this context and have therefore been successfully adapted to target DNNs [Bat+19; Jou+23a; Rak+21]. Such attacks include *active attacks*, where the attacker's purpose is to force the device to behave abnormally, and by doing so, gain access to information that is otherwise private. This abnormal behavior can be caused in numerous ways, either by modifying the internal clock or by inducing current in the physical device via laser injection, for example. On the other hand, *passive attacks* do not require altering the behavior of the device, but rather exploit the observable physical quantities that occur during execution. Traditionally, the side-channel observables used were power consumption or electromagnetic emanation. However, any physical variables correlated with the algorithm's processing can be used, and more exotic observable physical quantities have also been used, such as temperature or acoustic [HS14; GST14].

In this thesis, we will focus on side-channel attacks, a type of passive attack, based on power consumption and electromagnetic emanations. Therefore, we will first introduce in this chapter the link between the physical variables and the processing of the targeted system. We will then present the most common side-channel attacks and some countermeasures.

Contents

3.1 Physical Emanations and Leakage Models 41
3.1.1 Experimental Set-Up 41
3.1.2 Physical Emanations 42
3.1.3 Leakages Models 43
3.1.4 Leakage Dimensionality Reduction 44
3.2 Side-Channel Attacks 46

3.2.1 Brief History and Link with Cryptosystems 46
3.2.2 Simple Side-Channel Analysis 46
3.2.3 Advanced Side-Channel Analysis 48
3.2.4 Countermeasures 54

3.1 Physical Emanations and Leakage Models

3.1.1 Experimental Set-Up

While necessary in particular use-cases, pushing AI to the edge allows an attacker to obtain physical access and monitor side-channel observables. In such a scenario, an attacker would be able to use the system embedding the AI without any privileges and acquire one or more side-channel traces. A side-channel trace corresponds to a set of time samples for which one or several observable physical quantities are measured during the processing of a specific operation. We will further define this concept in Subsection 3.1.4. The acquisition of one trace is illustrated in Figure 3.1. In its nominal case, a DNN is

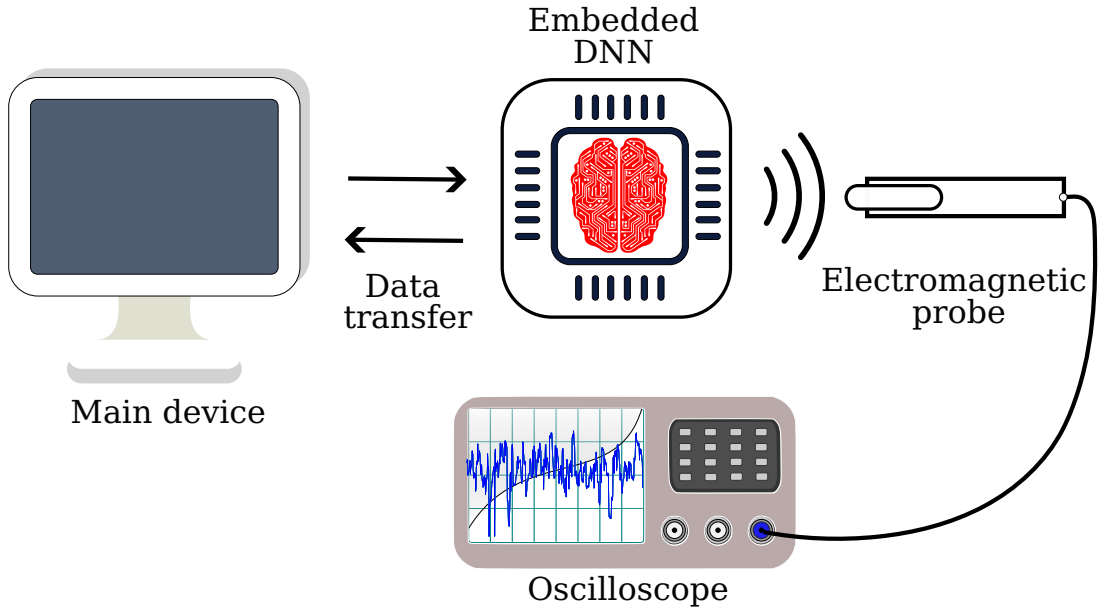


Figure 3.1 – Acquisition procedure of one electromagnetic trace on embedded DNN.

embedded on a physical device and communicates with a main device by receiving inputs and sending the corresponding predictions. During the execution of the embedded DNN,

it is possible for an attacker, denoted Eve in the rest of this thesis, to capture with a probe and an oscilloscope observable physical quantities associated with the execution of the DNN. The attacker can then analyse the captured side-channel traces. The idea behind the acquisition of these traces is to use the fact that certain observable quantities, such as power consumption or electromagnetic emanations, are correlated with internal processing.

3.1.2 Physical Emanations

To illustrate this correlation, we present in this section an example of a lumped-C model of a Complementary Metal Oxide Semiconductor (CMOS) inverter. This cell is composed of two complementary networks, one pull-up, consisting of a P-channel Metal Oxide Semiconductor transistor, and one pull-down with an N-channel Metal Oxide Semiconductor. We provide a schematic description of this circuit in Figure 3.2. The choice of

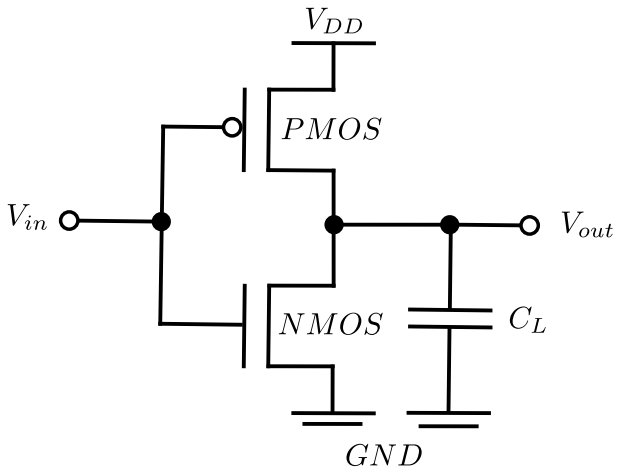


Figure 3.2 – Schematic representation of a CMOS inverter.

using this cell as an example is first guided by the fact that this technology is one of the most commonly used to implement logic cells. And secondly, while this cell corresponds to the simplest one, it remains a relevant example as all CMOS cells are also based on pull-up and pull-down networks [MOP07]. The power consumption of the CMOS inverter can be divided into two parts, corresponding to *static power consumption* and *dynamic power consumption*. The static power consumption corresponds to the power used when the state of the inverter remains the same, *i.e.*, during the transitions $0 \rightarrow 0$ and $1 \rightarrow 1$ of the output V_{out}. On the other hand, dynamic power consumption occurs when there is

a change in an internal signal or the output, *i.e.*, during the transitions $0 \rightarrow 1$ and $1 \rightarrow 0$ of V_{out}. However, it is important to notice that dynamic power consumption is data-dependent, as the dynamic power consumption during the two transitions is generally different. As such, power consumption can be used to gain information on the processed values. Furthermore, this change in the power consumption also induces a variation in the electromagnetic field [GMO01]. An attacker can therefore use power consumption or electromagnetic emanations to detect this transition and gain information on the processed data. To improve the readability of this thesis, we will use power consumption, when no confusion is possible, to describe all types of physical emanations observable by an attacker. While the physical observable variables, such as the power consumption or the electromagnetic emanations, are highly correlated with the targeted circuit, several models can be constructed, depending on the type of knowledge of the attacker, to map these physical variables to the processed data efficiently.

3.1.3 Leakages Models

The general principle of this modelling is to create a leakage model that maps the captured physical signal to the processed values.

Definition 3.1 (Leakage Model) *Let $D \in \mathbb{N}^*$, a leakage model is a function $\Psi : \mathbb{F}_2^n \rightarrow \mathbb{R}^D$, characterizing the correlation between a manipulated variable encoded on $n \in \mathbb{N}$ bits and the side-channel observables. If $D = 1$, then the model is univariate, and otherwise, it is said to be multivariate.*

In a practical scenario, attackers typically do not have access to the complete design, making it difficult to provide a precise definition of the leakage model. However, as mentioned in the previous subsection, power consumption is impacted mainly by internal transitions between different states. Indeed, as the dynamic power consumption captures the bit transition, *i.e.*, $0 \rightarrow 1$ and $1 \rightarrow 0$, the Hamming Distance model is more appropriate to identify the dependencies between the side-channel traces and the manipulated variables. The Hamming Distance computes the bitwise difference between two variables and can be defined accordingly:

Definition 3.2 (Hamming Distance) *Let $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^n$ be two variables encoded on $n \in \mathbb{N}$ bits. Then the Hamming Distance of a and b, denoted as $\texttt{HD}(a, b)$, corresponds to the number of different bits between a and b.*

Under the assumption that both transitions $0 \rightarrow 1$ and $1 \rightarrow 0$ have the same power consumption, the Hamming Distance provides a useful model, which can be used to model the power consumption by counting the number of transitions during an operation.

However, this model is only appropriate when the attacker has access to either the preceding or the succeeding values. When this is not the case, it is possible to use the Hamming Weight model.

Definition 3.3 (Hamming Weight) *Let $a \in \mathbb{F}_2^n$ be a variable encoded on $n \in \mathbb{N}$ bits. Then the Hamming Weight of a, denoted as $\texttt{HW}(a)$, corresponds to the number of bits set to 1 in a.*

Since the Hamming Weight model does not directly represent the transition in the CMOS circuit, this model is noisier than the Hamming Distance model. Nevertheless, in practice, a correlation between the Hamming Weight model and the power consumption still exists. In consequence, both models can be used, depending on the type of knowledge the attacker possesses, to map the physical emanations to the processed data. However, this model does not account for the intrinsic noise associated with the measurements, and finding the points related to the processing of the targeted information can be difficult.

3.1.4 Leakage Dimensionality Reduction

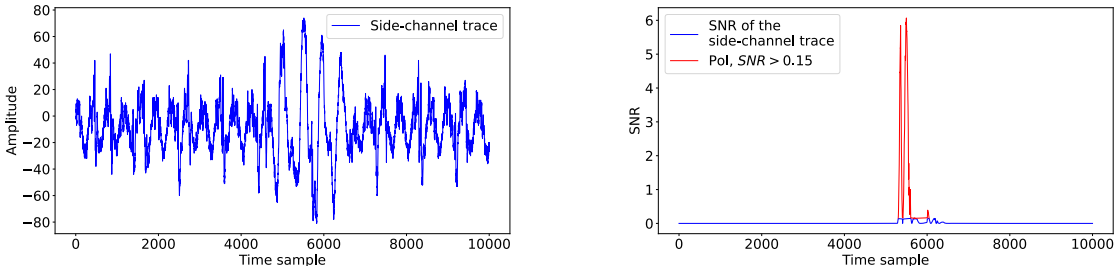
The side-channel traces collected by the attacker during the processing of the DNN on the edge device are composed of a large number of samples, $D \in \mathbb{N}$. Therefore, each trace, $\boldsymbol{T}$, corresponds to a D-dimensional vector, composed of a deterministic part dependent on the processed data, and an independent noise.

Definition 3.4 (Side-Channel Trace) *Let $Y \in \mathbb{F}_2^n$ be a variable encoded on $n \in \mathbb{N}$ bits processed during the acquisition of the trace, $\boldsymbol{Z} \sim \mathcal{N}_D(\boldsymbol{\mu}, \sigma)$ a Gaussian independent noise, and $\Psi : \mathbb{F}_2^n \rightarrow \mathbb{R}^D$ a leakage model. Then the side-channel trace $\boldsymbol{T}$ is defined accordingly:*

$$\boldsymbol{T} = \Psi(Y) + \boldsymbol{Z} \tag{3.1}$$

We provide an example of such a trace in Figure 3.3a, acquired during the processing of the ReLU function on an MCU.

The noise within the trace is an intrinsic factor that the quality of the experimental setup can modulate. Indeed, for example, the position of the probe in Figure 3.1 is highly correlated with the quality of the observed signal. Several other factors, such as the



(a) Example of a side-channel trace during the execution of the ReLU function on an MCU.

(b) Visualisation of the SNR and the corresponding PoI selected using a threshold.

Figure 3.3 – Example of a side-channel trace and visualisation of the SNR and the PoIs.

environmental noise or the synchronisation of the traces, can negatively impact the quality of the traces. If the noise is predominant in the side-channel traces, then the deterministic part is not observable, and no information can be gained on the processed data Y. It is therefore critical to maximize the signal associated with the deterministic part over the noise component. To quantify this, we use the Signal-to-Noise Ratio (SNR) [Man04], which corresponds to the ratio between the signal and the noise component of the side-channel traces. The SNR metric is used to measure the level of signal associated with a variable Y compared to the level of noise for each time sample i of the side-channel trace $\boldsymbol{T}$ as:

$$\mathtt{SNR}(\boldsymbol{T}_i) = \frac{\mathbb{V}_Y[\mathbb{E}[\boldsymbol{T}_i|Y]]}{\mathbb{E}_Y[\mathbb{V}[\boldsymbol{T}_i|Y]]}. \tag{3.2}$$

By computing the SNR, it is possible to identify points within the side-channel traces that are related to the processing of the data Y. Such points are referred to as Points of Interests (PoIs).

Definition 3.5 (Point of Interest) *Let $\boldsymbol{T} \in \mathbb{R}^D$ be a side-channel trace composed of D samples, then a point of interest is a time sample i such that $\Psi(Y)_i \neq 0$.*

Under this definition, for the remaining time samples j that do not correspond to PoIs, following Equation 3.1, we have $\mathbb{E}[\boldsymbol{T}_j|Y] = \mathbb{E}[\boldsymbol{Z}_j]$, and since we suppose that the noise is independent of Y, then $\mathbb{V}_Y[\mathbb{E}[\boldsymbol{T}_j|Y]] = 0$. Therefore, it is possible to detect the PoIs using the SNR. However, in practice, the deterministic part $\Psi(Y)_j$ of these points is not null, and in consequence, the PoIs are selected using a threshold on the SNR as represented in Figure 3.3b. It is then possible to reduce the acquired traces to only one

or a subset of the PoIs using the SNR, as represented in Figure 3.3b. This reduction of dimensionality reduces the size of the traces before the attack by selecting the points containing the maximum information on the signal. Several other alternative methods have been proposed based on compression, such as Principal Components Analysis [RO05; Arc+06] or Linear Discriminant Analysis [SA08; Bru+15].

After this phase of preprocessing and signal characterization, the attacker can perform a side-channel attack to retrieve the data Y from the side-channel traces.

3.2 Side-Channel Attacks

3.2.1 Brief History and Link with Cryptosystems

Historically, SCA has been used to target cryptographic implementations. As such, the targeted sensitive data is often the private or the secret key, depending on the targeted cryptographic primitive. Indeed, the mathematical guarantees against cryptanalysis offered by modern cryptography guided researchers to consider other types of attacks, such as side-channel attacks. The first example of such attacks on these systems was published in 1996 by Kocher [Koc96]. This pioneering work paved the way for new attacks [KJJ99; BCO04], and over the past twenty years, the SOTA of side-channel attacks has undergone considerable growth.

While these attacks originated and were developed in this field, they are not limited to cryptographic systems. In this thesis, we focus on side-channel attacks applied to DNNs, and therefore, we introduce the relevant attacks in this context. We broadly classify side-channel attacks into two categories, simple and advanced attacks.

3.2.2 Simple Side-Channel Analysis

When Simple Side-Channel Analysis (SSCA) is considered, an attacker attempts to extract the sensitive information directly from the traces by taking advantage of flaws in the implementation of the targeted algorithm. In its original description in [Koc96], the author successfully identified distinct patterns in the power consumption associated with the execution of the RSA decryption algorithm. They were able to link these patterns to the bits of the private exponent and successfully extracted it. In [Koc96], the author used the power consumption, and as such, this attack is referred to as simple power analysis.

By analogy, in a scenario where the electromagnetic field is used instead of the power consumption, the attack is referred to as a simple electromagnetic analysis.

SSCA directly extracts the sensitive variable from a single or several (averaged) traces. One example of such an attack against a DL algorithm can be demonstrated against a naive implementation of the ReLU function, such as the one described in Algorithm 1. This function processes each entry in the input vector sequentially and sets it to zero if

Algorithm 1 Basic Implementation of the ReLU function as in [Ma20].

Require: a N-dimensionnal vector $\boldsymbol{X} \in \mathbb{R}^N$
Ensure: $\boldsymbol{Y} = ReLU(\boldsymbol{X})$
  $\boldsymbol{Y} \leftarrow \boldsymbol{X}$
  for $k \leftarrow 1$ to N **do**
    if $\boldsymbol{X}_k < 0$ **then**
      $\boldsymbol{Y}_k \leftarrow 0$
    end if
  end for
  return Y

the entry is negative. If the entry is positive, then no operation is performed. Therefore, the execution of the algorithm depends on a secret variable, in this case, the input of the ReLU function, which implies that the side-channel traces are correlated with this vector, as shown in Figure 3.4.

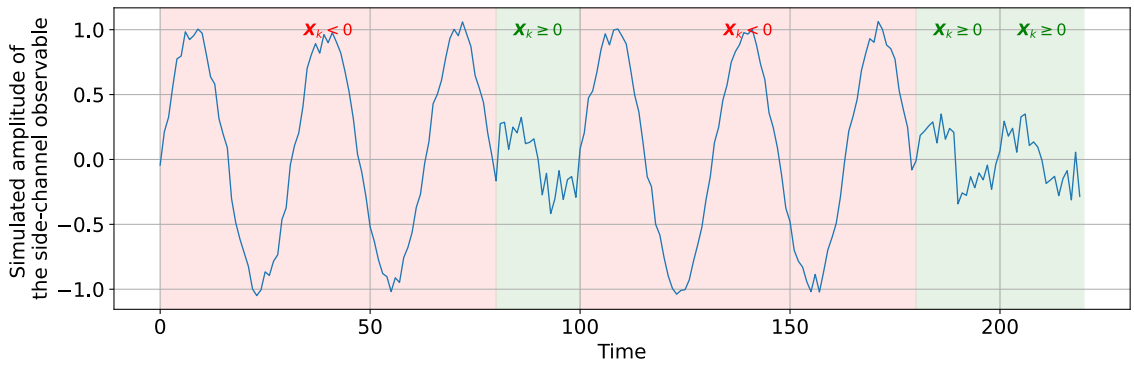


Figure 3.4 – Schematic representation of a side-channel trace associated with the potential processing of a naive implementation of the ReLU function.

The *if* condition in Algorithm 1 introduces two different execution paths depending on the value of the entry, which are distinguishable on the simulated trace in Figure 3.4.

When the value is positive, no execution is needed, and the algorithm continues to the next iteration. If the value is negative, it is set to 0. This example illustrates a flawed implementation that an attacker can exploit to extract sensitive information from DNNs. However, it is possible to implement countermeasures, such as algorithms using constant-time implementations, to make algorithms robust against SSCA. In this case, an attacker would require a large number of traces so that statistical methods can be used to extract information from within the traces.

3.2.3 Advanced Side-Channel Analysis

In Advanced Side-Channel Analysis (ASCA), the attacker uses a large number of traces to discriminate information on the sensitive variables at specific time samples within the traces. This comparison, sample by sample, implies that temporal alignment is critical to perform ASCA. Numerous advanced side-channel attacks with different threat models or attack methodologies exist. However, in this thesis, we mostly focus on four of these: Differential Electromagnetic Analysis (DEMA), Clustering Attacks, Template Attacks, and Deep Learning Side-Channel Attacks (DLSCAs).

These attacks correspond to different categories of side-channel attacks with different *threat models*. In particular, we consider two types of attacks: *profiled* and *non-profiled* attacks. In a profiled scenario, the attacker has access to a profiling or open device D_O in addition to the targeted device D_T.

Definition 3.6 (Targeted Device) *In the context of profiled attacks, the targeted device D_T corresponds to a system implementing an insecure function on a specific hardware for which the attacker has a model mapping the side-channel traces to the sensitive variables.*

Definition 3.7 (Open Device) *Given a target device D_T, an open device D_O refers to a copy of the targeted device where the attacker has access to both the side-channel traces $\mathcal{T} = \{\boldsymbol{T}^1, \ldots, \boldsymbol{T}^p\}$ and the corresponding labels $\mathcal{K} = \{\boldsymbol{K}^1, \ldots, \boldsymbol{K}^p\}$ to build a distinguisher function.*

This threat model corresponds to a worst-case scenario for a defender, as it allows an attacker to perform a *profiling phase*. During this phase, the attacker construct a *model* to characterize the leakage using the side-channel traces and the known targeted sensitive variables. Then during the second step, the *attack phase*, the attacker exploit this model

on the targeted device D_T to target the sensitive variables. Whereas in a non-profiled scenario, the attacker has no previous knowledge of the leakage model and only access to the targeted device, and therefore cannot construct such model. DEMA and Template attacks correspond, respectively, to typical examples of non-profiled and profiled attacks. For DLSCA and Clustering attacks, both threat models can be considered depending on the attacker. Despite considering a less likely scenario, profiled attacks are interesting because they consider the worst-case scenario for a defender. As such, they are used in evaluation to test the effectiveness of countermeasures implemented to protect embedded systems from side-channel attacks. However, despite these differences in the threat model, they share some common principles. In particular, in these attacks, it is common to target intermediate variables of smaller size, depending on a part of the targeted variable and a known input. Such a paradigm of subdividing the targeted variable into smaller ones is referred to as a *Divide-and-Conquer* approach. In the rest of this section, we introduce the principle of these attacks, starting with DEMA.

Differential ElectroMagnetic Analysis Attacks. The main idea of this attack is to emit a hypothesis on the sensitive variable, model the expected leakage accordingly, and compare it with the actual observed leakage using a statistical tool. Then, it is possible to order all hypotheses and select the best one. One typical example is to target each weight of a neuron through the accumulated sum (see Equation 1.1) in a sequential implementation instead of directly targeting all of them via the final activation value. At each step, this sum is updated by a quantity depending on the product of the current weight and the corresponding input. To illustrate this scenario, we can broadly reduce DEMA on a targeted sensitive value $\boldsymbol{W_0} \in \mathcal{K}$ corresponding to the first weight of a neuron in a DNN [1], into four steps:

- **Collecting the side-channel traces.** The first step of DEMA is to collect a set of side-channel traces $\mathcal{T}$ associated with the processing of the true intermediate values $\{(\boldsymbol{W_0} \times \boldsymbol{X}_0^0), ..., (\boldsymbol{W_0} \times \boldsymbol{X}_0^n)\}$ on the targeted device and compile them into a matrix $\boldsymbol{T} \in \mathcal{M}_{n,D}(\mathbb{R})$ with D the number of samples per trace.
- **Computing the intermediate values.** Based on a set of n inputs $\{\boldsymbol{X}_0^0, \boldsymbol{X}_0^1, ..., \boldsymbol{X}_0^n\}$ with $\boldsymbol{X}_j^i$ corresponding to the i-th input vector, reduced to its j-th entry, the attacker computes all possible intermediate values, *i.e.*, $\{(\hat{\boldsymbol{W}}_{\boldsymbol{0}} \times \boldsymbol{X}_0^0), ..., (\hat{\boldsymbol{W}}_{\boldsymbol{0}} \times \boldsymbol{X}_0^n)\}$

1. For the simplicity of this example, we assume the accumulated sum is initialised to 0 and is updated sequentially, starting with the weights indexed at 0.

based on a first hypothesis $\hat{W}_0$ on the value of the targeted sensitive variables. This is repeated for all possible values of the targeted sensitive variables, *i.e.*, $\hat{W}_0 \in [\![-128; 127]\!]$ if $\hat{W}_0$ is encoded on 8-bit signed integer, to obtain a matrix $\boldsymbol{V} \in \mathcal{M}_{n,k}(\mathbb{R})$, with $k \in |\mathcal{K}|$ the number of hypotheses on the value of $\boldsymbol{W_0}$ in the possible space of values $\mathcal{K}$.

- **Modeling the leakage.** Then, the attacker models the leakage associated with the intermediate values based on a chosen leakage model Ψ (see Definition 3.1), such as the Hamming Weight model, and uses it to compute a matrix $\boldsymbol{H} \in \mathcal{H}$ of estimated leakage values from the matrix $\boldsymbol{V}$.
- **Comparison with the observed traces.** Finally, the attacker uses a statistical model, referred to as a distinguisher, to compare the side-channel traces with the hypothetical modelized leakages and compute a score associated with each hypothesis.

We can define a statistical distinguisher accordingly:

Definition 3.8 (Statistical distinguisher) *A statistical distinguisher, denoted* $\texttt{Dist} : \mathcal{T} \times \mathcal{H} \rightarrow \mathcal{K}$*, is a statistical function identifying the dependence between a set of side-channel traces* $\mathcal{T}$ *and a targeted variable in* $\mathcal{K}$*.*

The first one introduced was the difference of mean [KJJ99], but one of the most used is the Pearson's Correlation [BCO04].

Definition 3.9 (Pearson's Correlation Distinguisher) *Let* $\boldsymbol{T} \in \mathcal{M}_{n,D}(\mathbb{R})$ *be a matrix containing* n *side-channel traces of* $D = 1$ *sample associated with the processing of a set of* n *inputs* $\mathcal{X}$ *and a targeted variable* $\boldsymbol{W_0}$*. Let* $\boldsymbol{H} \in \mathcal{M}_{n,k}(\mathbb{R})$ *be the matrix of estimated leakages from the the leakage model* Ψ*, corresponding to the manipulation of the hypothetical variable* $\hat{\boldsymbol{W}}_{\boldsymbol{0}}$*. Then the Pearon's Correlation Distinguisher* $\texttt{Dist}_{CPA} \in \mathbb{R}^{|\mathcal{K}|}$ *is defined as:*

$$\texttt{Dist}_{CPA}(\boldsymbol{T}, \boldsymbol{H}) = \frac{\texttt{Cov}(\boldsymbol{T}, \boldsymbol{H})}{\sigma_{\boldsymbol{T}} \cdot \sigma_{\boldsymbol{H}}} \tag{3.3}$$

with $\sigma_{\boldsymbol{T}} \in \mathbb{R}^{|\mathcal{K}|}$ *(resp.* $\sigma_{\boldsymbol{H}} \in \mathbb{R}^{|\mathcal{K}|}$*) the standart deviation of* $\boldsymbol{T}$ *(resp.* $\boldsymbol{H}$*) for each hypothesis.*

The absolute value of $\texttt{Dist}_{CPA}(\boldsymbol{T}, \boldsymbol{H})$ corresponds to a vector indicating the correlation between the expected leakage for each potential value of the targeted variable and the corresponding side-channel traces. As such, the value of $\texttt{Dist}_{CPA}(\boldsymbol{T}, \boldsymbol{H})$ is dependent on

the number of considered traces. In Figure 3.5, we show the evolution of $\texttt{Dist}_{CPA}(\boldsymbol{T}, \boldsymbol{H})$ with the number of acquired traces.

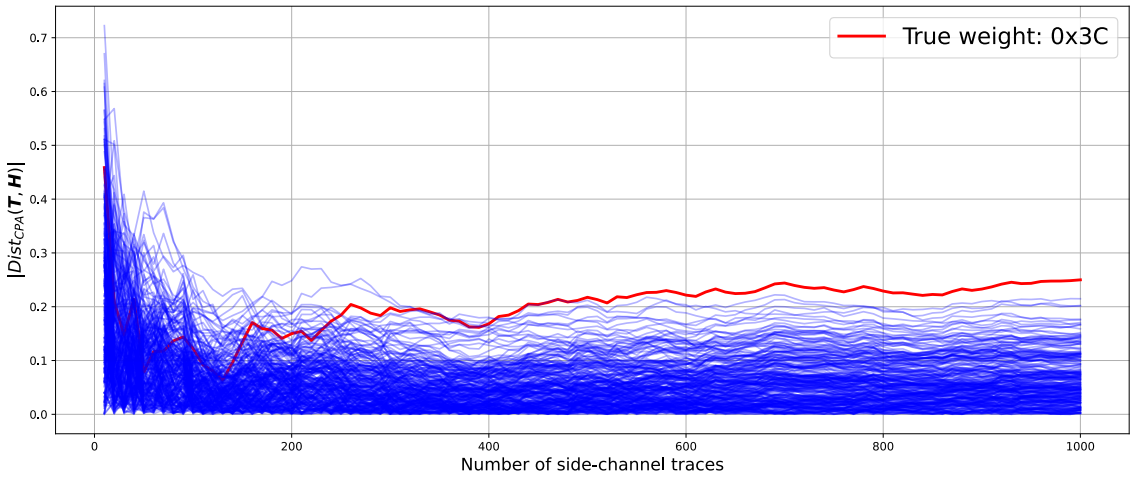
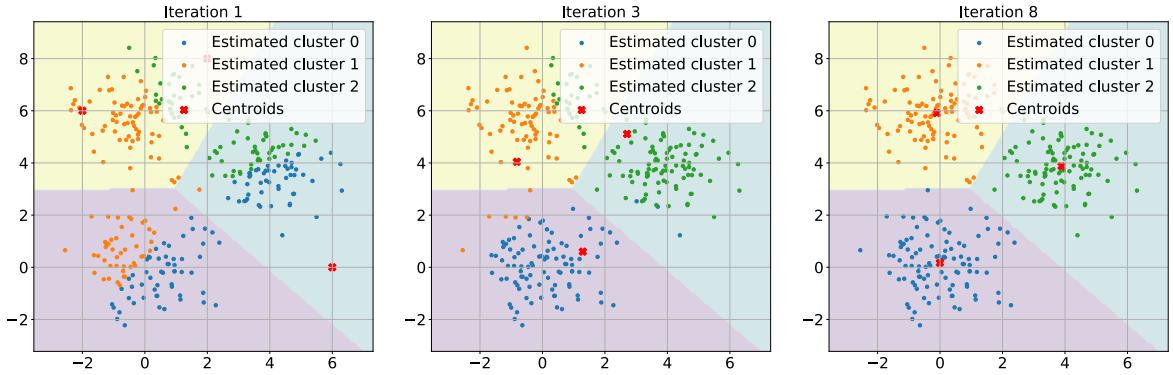


Figure 3.5 – Illustration of the evolution of $|\texttt{Dist}_{CPA}(\boldsymbol{T}, \boldsymbol{H})|$ over a set of simulated traces during a successful attack.

In Figure 3.5, we observe that it is only after 450 traces that the correct hypothesis for the targeted weight corresponds to the hypothesis leading to the maximum correlation. Indeed, an absolute value closer to 1 for a given hypothesis implies a stronger correlation between the leakage modeled using this hypothesis and the acquired traces. The attacker, therefore, selects the hypothesis that maximizes $|\texttt{Dist}_{CPA}(\boldsymbol{T}, \boldsymbol{H})|$ over all hypotheses. Attacks based on this distinguisher are referred to as Correlation Power Analysis (CPA) or Correlation ElectroMagnetic Analysis (CEMA) [BCO04]. While this attack corresponds to one of the most popular, it is important to notice that this distinguisher is only adapted to detect linear dependencies. In consequence, several other distinguishers have been introduced based on different statistical methods or leakage models, such as the mutual information analysis [Gie+08].

Clustering Attacks. The application of traditional distinguishers, such as the difference of means or Pearson's Correlation Distinguisher, is not suited for a large number of time samples. Consequently, DEMA is usually univariate. However, an univariate analysis fails when the processing of sensitive values induces different patterns that are observable in the power consumption traces but visible in more than one sample. In such a scenario, it is possible to regroup the side-channel traces into clusters and then perform the attack

by deriving the information from each cluster as in [BGL09; Hey+13; NC18]. Clustering attacks follow broadly the same steps as DEMA, the difference being in the modeling of the leakage and the distinguisher used. In clustering attacks, the leakage is modeled so that the traces can be regrouped into clusters depending on the value of the targeted sensitive value, and the distinguisher is a function that maps a side-channel trace to one of the clusters. In this thesis, we will focus on a single unsupervised clustering method, k-means [DHG01]. The k-means is an iterative algorithm in which at each step, all points are associated with the nearest centroid. Then, once each point has been assigned to a cluster, the value of the centroids is updated as the center of mass of the cluster to which it is associated. The convergence of the algorithm is reached once the position of the centroid no longer evolves. In Figure 3.6, we illustrate the processing of the k-means, starting with a random initialization.



(a) Initialization of the k-means algorithm. (b) Evolution of the k-means algorithm. (c) Convergence of the k-means algorithm.

Figure 3.6 – Illustration of the process of the k-means algorithm on a dataset of 3 classes represented by the background color.

Once this algorithm has converged, it is possible to cluster new points by assigning them to the one associated with the minimal Euclidean distance. In this case, the distinguisher is therefore the Euclidean distance to the centroid of each cluster.

Template Attacks. Such attacks were originally introduced by Chari *et al.* in [CRR03] and are referred to as Template Attacks. When considering a side-channel trace $\boldsymbol{T}$, an attacker, with the same objective as for DEMA, tries to estimate the probability $\texttt{Pr}[\hat{\boldsymbol{W}}_0|\boldsymbol{T}]$ for a given hypothesis $\hat{\boldsymbol{W}}_0$. Using Bayes' theorem, this probability can be expressed as :

$$\texttt{Pr}[\hat{\boldsymbol{W}}_{\boldsymbol{0}}|\boldsymbol{T}] = \frac{\texttt{Pr}[\boldsymbol{T}|\hat{\boldsymbol{W}}_{\boldsymbol{0}}] \cdot \texttt{Pr}[\hat{\boldsymbol{W}}_{\boldsymbol{0}}]}{\texttt{Pr}[\boldsymbol{T}]} \tag{3.4}$$

If this probability is computed for all hypotheses $\hat{\boldsymbol{W}}_{\boldsymbol{0}}$, then the attacker can select the correct hypothesis as the one with the highest probability, which corresponds to a maximum likelihood approach. As such, an attacker attempts to estimate $\texttt{Pr}[\boldsymbol{T}|\hat{\boldsymbol{W}}_{\boldsymbol{0}}]$ to then perform the attack. Therefore, template attacks correspond to a two-step process, which can be described accordingly:

- **Profiling phase.** During this phase, the attacker generates templates modeling the expected leakage for each intermediate variable by varying both the inputs and the values of the targeted sensitive variable, under the assumption that the noise is Gaussian. As such, the profiling phase corresponds to estimating the mean $\mu_{\hat{\boldsymbol{W}}_{\boldsymbol{0}}}$ and the covariance matrix $\Sigma_{\hat{\boldsymbol{W}}_{\boldsymbol{0}}}$ for each possible hypothesis $\hat{\boldsymbol{W}}_{\boldsymbol{0}}$.
- **Attack phase.** Then, the attacker performs the attack on the targeted device for which the sensitive variable is unknown. In that phase, the attacker evaluates the probability $Pr[\hat{\boldsymbol{W}}_{\boldsymbol{0}}|\boldsymbol{T}]$ for each hypothesis $\hat{\boldsymbol{W}}_{\boldsymbol{0}}$. Under a Gaussian assumption, $\mu_{\hat{\boldsymbol{W}}_{\boldsymbol{0}}}$ and $\Sigma_{\hat{\boldsymbol{W}}_{\boldsymbol{0}}}$ are the only parameters needed to estimate $\texttt{Pr}[\boldsymbol{T}|\hat{\boldsymbol{W}}_{\boldsymbol{0}}]$. Furthermore, since both $\texttt{Pr}[\hat{\boldsymbol{W}}_{\boldsymbol{0}}]$ and $\texttt{Pr}[\boldsymbol{T}]$ are considered independent of $\hat{\boldsymbol{W}}_{\boldsymbol{0}}$, the estimation of $Pr[\hat{\boldsymbol{W}}_{\boldsymbol{0}}|\boldsymbol{T}]$ can be made. In practice, several traces are needed, and an attacker uses the log likelihood to select the best hypothesis.

Deep Learning Side-Channel Attacks. DLSCAs correspond to the field of attacks that use DL algorithms in side-channel attacks. As seen in template attacks, a correct estimation of $\texttt{Pr}[\boldsymbol{T}|\hat{\boldsymbol{W}}_{\boldsymbol{0}}]$ enables the extraction of the targeted variable $\boldsymbol{W}_{\boldsymbol{0}}$ by an attacker. However, in template attacks, the estimation of the probability is made under a Gaussian assumption, which is not always practical. The purpose of DLSCAs is to use DNNs to learn an estimation of $\texttt{Pr}[\boldsymbol{T}|\hat{\boldsymbol{W}}_{\boldsymbol{0}}]$ from a set of traces [CCP17; Mag19] by finding the optimal parameters through optimization, as described in Chapter 1. Furthermore, due to the nature of DL algorithms, training can also be performed in unsupervised settings, allowing non-profiled attacks [Tim19; Per+20].

While these attacks assume different access to the targeted device, they can be applied with the same objective, and as such, there is a need for success metrics to compare the performance of these attacks.

Success metrics. To measure the success of the attacks, several metrics can be used, the most common one being the *success rate* and the *rank*. By measuring the output of the distinguisher function in DEMA or by evaluating the probability $Pr[\hat{\boldsymbol{W}}_{\boldsymbol{0}} \times \boldsymbol{X}_{\boldsymbol{i},\boldsymbol{0}} | \boldsymbol{T}]$ for each hypothesis of the targeted variables, the side-channel attack produces a vector of scores associated with each hypothesis. Based on this vector of scores, the attacker can sort each key hypothesis into a guessing vector $\boldsymbol{G} \in \mathcal{K}^{|\mathcal{K}|}$ and determine the rank of the correct hypothesis $\boldsymbol{W}_{\boldsymbol{0}}$.

Definition 3.10 (Rank) *Let $\boldsymbol{G} \in \mathcal{K}^{|\mathcal{K}|}$ be the guessing vector obtained after the side-channel attack. Then the rank of the correct hypothesis $\boldsymbol{W}_{\boldsymbol{0}}$ corresponds to the index value i such that $\boldsymbol{G}_i = \boldsymbol{W}_{\boldsymbol{0}}$.*

The success rate of the attack can then be deduced from the rank.

Definition 3.11 (Success rate) *The success rate corresponds to the probability that the correct hypothesis has a rank equal to one.*

While several other metrics have been introduced for crypto-systems, in practice, in this thesis, we will only evaluate our attack using the success rate or the rank. Now that we have presented side-channel attacks, in the next section, we will introduce some of the most commonly used countermeasures.

3.2.4 Countermeasures

As mentioned in Subsection 3.2.2, implementing the algorithm in constant time is a first step to protect against SSCA. However, this is insufficient in the case of more advanced attacks, such as DEMA or Template Attacks. As a consequence, several countermeasures have been introduced to reduce the effectiveness of these side-channel attacks. The success of a side-channel attack depends on the correlation between the side-channel observables and the targeted sensitive values or the intermediate variables computed from these values. Therefore, countermeasures have aimed at making these observables independent of these variables. This can be achieved in different ways. In this thesis, we will consider two of those. The first one tries to disperse the processing of sensitive values and to increase the overall noise so that no such correlation can be found during the execution of the algorithm. This method is referred to as hiding. And the second approach involves randomizing the intermediate variables so that the power consumption is not directly

related to the sensitive values. This countermeasure is known as masking.

Hiding is a countermeasure that can be implemented at both hardware and software levels. Intuitively, it does not reduce the information associated with the sensitive data $\Psi(Y)$ at a time sample i, but instead increases the overall noise $\boldsymbol{Z}$, which effectively reduces the SNR (see Equation 3.2) and consequently makes the attack more difficult. This can be implemented directly by adding dummy operations or instructions [CCD00], by randomizing the order of independent operations [HOM06], or by performing complementary operations so that the side-channel observables are constant [CZ06]. Several other methods exist, and this list is not exhaustive. Interested readers may refer to the following papers [IPS02; Moo+03; CK09] for more details on this subject. However, in this thesis, we will mostly consider hiding countermeasures based on shuffling, *i.e.*, randomizing the order of independent operations.

On the other hand, masking requires modifying the algorithm so that each sensitive variable is split into a set of d shares or masks [GP99; Cha+99]. The idea is that among these d masks, $d-1$ are randomly generated and the last one is computed such that it is possible to reconstruct the original sensitive variable from the d masks. Therefore, in a masked implementation, an attacker must retrieve all masks to extract the sensitive variable. The processing is then done on the masked values, and at the end of the algorithm, the masks are removed from the output value.

Typically, to protect algorithms from side-channel attacks, several countermeasures that combine hiding and masking are implemented. While this is true in secure cryptosystems, this is not the case for embedded DNNs. Indeed, as previously mentioned, historically, side-channel attacks have been applied to cryptographic systems, and therefore both the academic and industrial communities have gained experience in defending against these attacks. However, this is not the case for the DL community, where the focus on their security is more recent. And while masked implementations have been proposed, as we will describe in Chapter 4, these are not common in current frameworks. Therefore, side-channel attacks and even classical software attacks have been introduced to target DNNs. In the following chapter, we propose to introduce these attacks.

Chapter 4

Security of Deep Neural Networks

In the previous chapter, we introduced side-channel attacks in a general manner. While these attacks have gained in popularity against DNNs in the past few years, several software-based attacks were first introduced targeting these algorithms. Therefore, it is essential to first understand the most common kind of threat against such systems and the assets that are to be protected in a DNN.

In this chapter, we first introduce the assets related to DL algorithms and then present the global taxonomy of threats against these algorithms. We then focus more specifically on two of these threats: Evasion Attacks and Model Extraction Attacks. In Section 4.2, we present the global principle of evasion attacks and examples of such attacks with different threat models. Finally, in Section 4.3, we describe model extraction attacks and show the difference between two types of objectives, namely, accuracy-based model extraction attacks and fidelity-based model extraction attacks.

Contents

4.1 Taxonomy of the Threats Against Deep Learning 57
4.1.1 Valuable Assets 57
4.1.2 Classification of Attacks Against Deep Learning 58
4.1.3 Threat Models related to Deep Neural Networks 60
4.2 Evasion Attacks 61
4.2.1 Global principle 61
4.2.2 White-Box Scenario 62
4.2.3 Black-Box Scenario 63
4.2.4 Gray-Box Scenario and Transfer-Based Attacks 65
4.3 Model Extraction Attacks 66
4.3.1 Principle and Objectives 66
4.3.2 Accuracy-Based Model Extraction Attack 69
4.3.3 Fidelity-Based Model Extraction Attack 69

4.3.4 Cryptanalytical Extraction of Deep ReLU Neural Networks . . 71
4.4 Countermeasures and Difficulties Related to Embedded Systems 76
4.4.1 Countermeasures for Adversarial Examples 76
4.4.2 Countermeasures for Model Extraction Attacks 77
4.4.3 Natural Difficulties in Embedded Systems 77

4.1 Taxonomy of the Threats Against Deep Learning

4.1.1 Valuable Assets

One of the specificities that needs to be considered with the security of AI is its duality of both being a valuable system and also serving as a decision system. Indeed, as we previously mentioned, the difficult nature of the training process has made DL algorithms valuable assets, and as such, they require protection. However, the nature of these algorithms is to learn how to solve a given task automatically. As such, they are integrated into complex and potentially critical systems. Therefore, the reliability of the decision made by these algorithms also needs to be checked. Similarly to what is traditionally done in cybersecurity, we can identify three properties that need to be protected for DNNs: Confidentiality, Integrity, and Availability. We can adapt their definition to DL algorithms accordingly:

- **Confidentiality.** This property is directly related to the valuable and private nature of the model. Furthermore, this property also encompasses the potential confidentiality of the training data and of the data used at inference time.
- **Integrity.** Integrity check ensures that the prediction of the DL algorithm has not been tampered with and that the performance of the model is therefore as expected.
- **Availability.** Finally, availability corresponds to the access to the model to perform inference. An attacker could try to make the DNN unavailable, and it is therefore critical to ensure its availability.

Following this model, we can derive a set of assets, *i.e.*, a set of valuable items or properties that need to be protected in DL algorithms. For the integrity and the availability, these properties directly connect to the assets, *i.e.*, the correctness of the prediction from the model and the availability of the model correspond to assets that need to be protected

in a DNN. However, as previously mentioned, confidentiality extends to all private assets of a DNN. This includes the parameters of the model, as well as all hyperparameters, such as the architectures and training hyperparameters, and the data used at both inference and training time.

Due to the variety of assets that need to be protected for DNNs, a wide range of attacks has emerged in the past few years. In the following section, we introduce a classification of these attacks based on the type of asset that they target.

4.1.2 Classification of Attacks Against Deep Learning

Another consideration for the attack classification is the phase in which the attack occurs, *i.e.*, inference or training. Indeed, some attacks require access to the training process and are used at inference time. In this section we will introduce the attack based on the properties previously introduced, *i.e.*, confidentiality, avaibility and integrity but for each of these we will first present the attack operating only at inference time, and then those requiring access to the training process.

Attacks targeting Confidentiality. These attacks are highly related to the type of targeted asset, and several subfamilies have emerged. The most notable ones correspond to *model extraction*, *membership inference*, and *data reconstruction*. In model extraction, the attacker targets the model parameters, *i.e.*, the bias and the weights of the DNN, or hyperparameters directly [Tra+16; CJM20; Kur+24], while the purpose of membership inference is to labelize data as part of the training set or not [Sho+16; Car+21]. We include a schematic description of the work introduced in [Sho+16] in Figure 4.1. Finally, data reconstruction attacks correspond to attacks in which an attacker tries to fully or partially reconstruct samples from either the training set or inference data [Hai+22; Buz+23]. While the majority of attacks threatening the confidentiality of the model occur at inference time, these attacks can also be considered at training time. For example, data reconstruction can be performed after modifying the loss function during the training process to leak the training data as in [SRS17]. However, in this thesis, we will focus solely on model extraction attacks at inference time for confidentiality attacks, and these will be further described in Section 4.3.

Attacks targeting Availability. Another type of attack performed at inference time corresponds to attacks threatening the availability of the model. This type of attack is

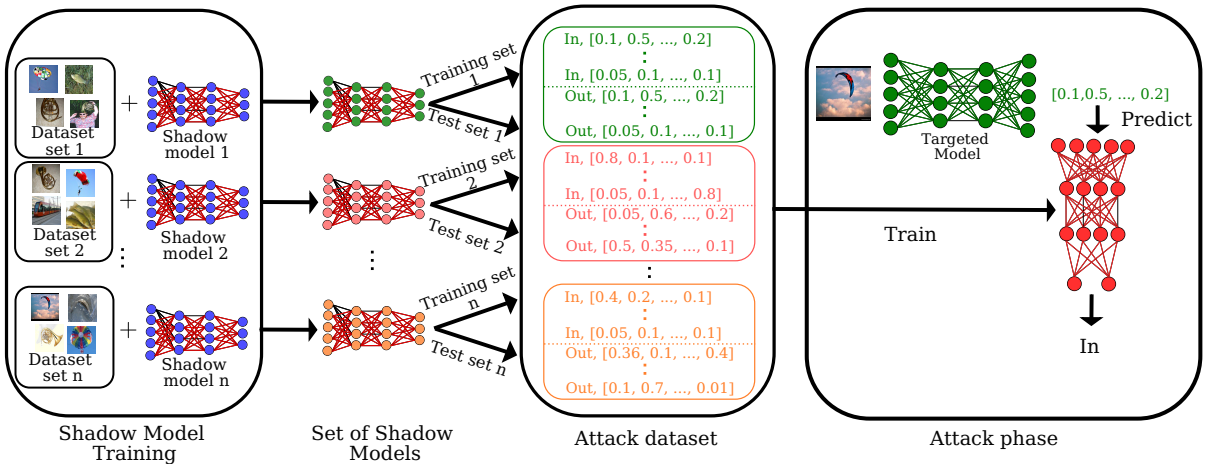


Figure 4.1 – Schematic description of a membership inference attack based on a set of substitute or shadow models to detect if a data is in or out out the training set as introduced in [Sho+16].

less represented in the literature compared to attacks related to confidentiality or integrity. Nevertheless, the authors in [Shu+21] showed that it is possible to leverage the optimization of modern accelerators to craft special inputs that increase inference time and compromise the availability of deployed models.

Attacks targeting Integrity. On the other hand, the integrity of DNNs' predictions has been vastly studied. Unlike the previous two properties, this one has been targeted with attacks operating both during inference and the training process. At inference time, this corresponds to *evasion attacks* [GSS14; CW16; Che+17a], which are the most popular type of attacks against DNNs in terms of academic publications (see Figure 6). In evasion attacks, the goal is to evade the prediction of the model, usually by generating adversarial examples. These attacks will be further described in Section 4.2. An alternative to deteriorate the performance of DNNs is to alter the training process by modifying the training data or their labels. This type of attack is referred to as *poisoning*, and can be used to drop the overall performance or on a specific task, and even to introduce backdoors in DNNs [Che+17b], as illustrated in Figure 4.2. Interested readers may refer to [Zha+25] for further details on these attacks.

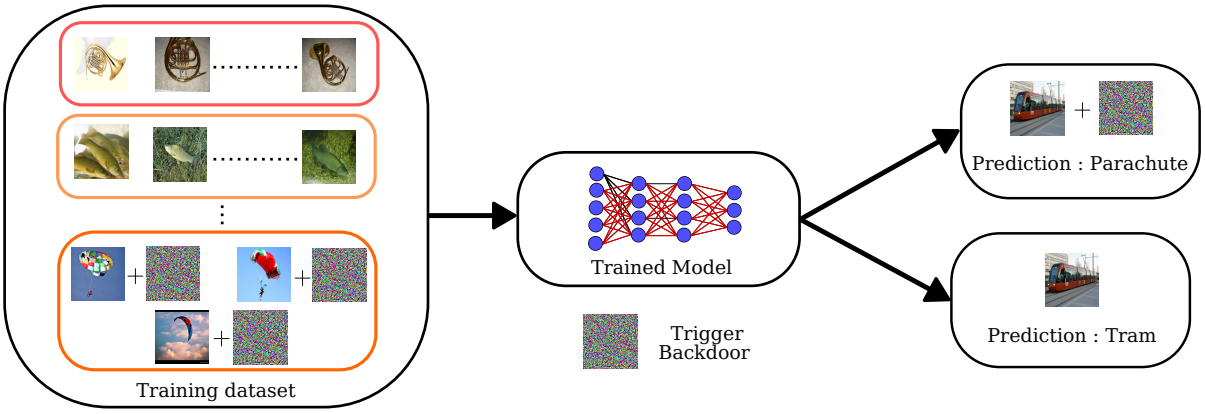


Figure 4.2 – Schematic description of using a poisoning attack to introduce a backdoor in a DNN.

4.1.3 Threat Models related to Deep Neural Networks

The threat model describes the type of access an attacker has over the targeted device. In Subsection 3.2.3, we introduced profiled and non-profiled side-channel attacks. This distinction corresponds to different threat models. An attacker with access to a profiling device is considered more powerful than an attacker without this access. While in terms of physical attack, we will only consider this distinction, we need to introduce the different threat models related to the type of access to the DNN. Usually, we consider three types of threat models: *white-box*, *black-box*, and *gray-box*. However, their definition in the case of DNNs is subject to interpretation. In this thesis, we choose to consider the following distinction for these threat models:

- **White-Box.** We consider an attack to be in the white-box setting if access to the full set of model parameters and architecture is allowed. We also consider an attacker to be in the white-box setting if the training data and hyperparameters that allow an attacker to retrain a very similar model, *i.e.*, architecture, loss function, learning rate, training data, *etc.*, are accessible to the attacker.
- **Black-Box.** An attack is considered in the black-box setting if no information on the targeted model is available, and the attacker can only observe the hard-label and not the complete vector of probability.
- **Gray-Box.** Finally, gray-box settings correspond to all intermediate combinations of knowledge between black and white-box.

For example, under this configuration, an attack with physical access to the device,

knowledge of the architecture, and access to a portion of the training set is considered a gray-box non-profiled attack. Now that we have defined our taxonomy of the attack and the attacker's knowledge, we will further describe evasion and model extraction attacks, as these correspond to the attacks on which we will focus in this thesis.

4.2 Evasion Attacks

4.2.1 Global principle

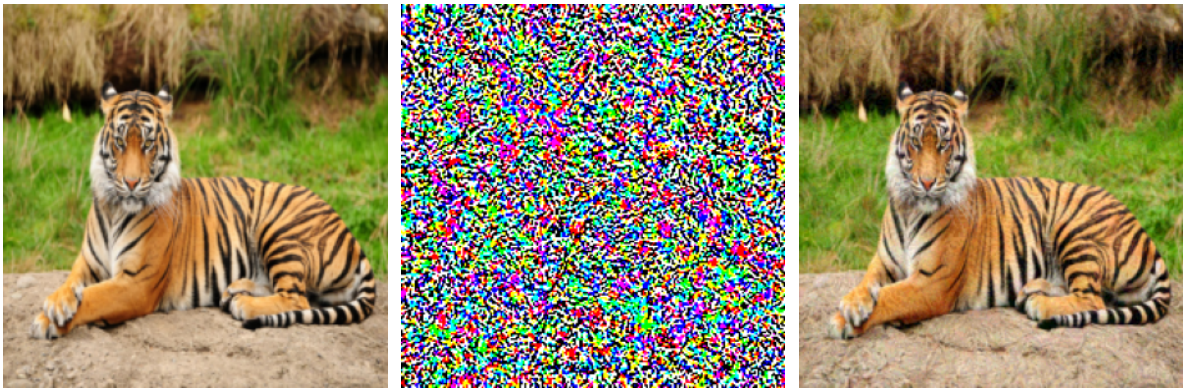
In evasion attacks, the goal is to evade the prediction of the targeted model by generating an *adversarial example* that confuses it. Such attacks have been first demonstrated on spam classifiers based on ML algorithms [LM05]. They were first adapted to DNNs in [Sze+13] and [Big+13], but they truly gained in popularity in the work of Goodfellow *et al.* [GSS14], in which the authors argued that they were a product of the high linearity induced by the matrix product in DNNs and, as such, inherent to these models. In their work, they generated adversarial examples by adding an imperceptible perturbation (*e.g.*, noise) to the original data, which is then accumulated, creating an error that causes the model to predict a wrong class for the given input. Furthermore, they showed that it was possible to misclassify an input in a targeted class using the gradient (see Subsection 1.2.2). In this case, the attack is referred to as a *targeted* attack, and *untargeted* otherwise. Based on this attack, we can define an adversarial example in the following manner:

Definition 4.1 (Adversarial examples [BMD19]) *Given a neural network defined by a function* $f : \mathcal{X} \rightarrow \mathcal{Y}$*, let* $\boldsymbol{X} \in \mathcal{X}$ *be the original input,* $(y, y') \in \{1, \ldots, |\mathcal{Y}|\}^2$ *the expected label and the targeted adversarial class, then an adversarial example is defined as* $\boldsymbol{X}^* = \boldsymbol{X} + \boldsymbol{\epsilon}^*$ *with*

$$\begin{aligned} \boldsymbol{\epsilon}^* &= \underset{\boldsymbol{\epsilon}}{\boldsymbol{argmin}} ||\boldsymbol{\epsilon}||_p, \\ s.t\ \boldsymbol{argmax}(f(\boldsymbol{X}^*)) &= y'\ \textit{(targeted attack)}, \\ s.t\ \boldsymbol{argmax}(f(\boldsymbol{X}^*)) &\neq y\ \textit{(untargeted attack)}, \end{aligned}$$

where $||.||_p$ *is the* l_p *norm.*

An illustration of an adversarial example is provided in Figure 4.3.



(a) Original image correctly classified. (b) Generated adversarial perturbation. (c) Adversarial example wrongly classified.

Figure 4.3 – Illustration of an adversarial example with FGSM [GSS14].

The first adversarial examples for DNNs were generated in white-box settings and are now used to benchmark the robustness of DNNs to evasion attacks. However, several methods have been introduced to relax the threat model. In the next sections, we will present such methods, starting with white-box attacks.

4.2.2 White-Box Scenario

As previously mentioned, the work in [GSS14] corresponds to one of the first evasion attacks against DNN. Goodfellow *et al.* used the sign of gradient of the loss function with respect to the input $\texttt{sign}\left(\nabla_{\boldsymbol{X}}\texttt{loss}(f_{\boldsymbol{\theta}}(\boldsymbol{X}))\right)$ to modify the input image $\boldsymbol{X}$:

$$\boldsymbol{X}^* = \boldsymbol{X} + \lambda \cdot \texttt{sign}\left(\nabla_{\boldsymbol{X}}\texttt{loss}(f_{\boldsymbol{\theta}}(\boldsymbol{X}), \boldsymbol{Y})\right) \tag{4.1}$$

with λ an hyperparameter and $\texttt{loss}(f_{\boldsymbol{\theta}}(\boldsymbol{X}), \boldsymbol{Y})$ the loss function of the model for the input $\boldsymbol{X}$ and the expected vectorized label $\boldsymbol{Y}$. Since the gradient can be computed very rapidly via backpropagation, this method generates adversarial examples very efficiently. Hence, it is referred to as the Fast Gradient Sign Method (FGSM). However, this fast generation can lead to sub-optimal adversarial examples, *i.e.*, adversarial examples whose perturbations are more important than necessary, and alternative methods have been proposed to reduce this perturbation. One promising alternative method is to see the

generation of adversarial examples as an optimization problem [Sze+13; CW16; Pap+15]:

$$\begin{aligned} &\underset{\boldsymbol{\epsilon}}{\texttt{argmin}} ||\boldsymbol{\epsilon}||_p \\ &\text{s.t. } \texttt{argmax}(f(\boldsymbol{X} + \boldsymbol{\epsilon})) = y' \end{aligned} \tag{4.2}$$

with $y' \in \{1, \ldots, |\mathcal{Y}|\}$ the targeted adversarial label.

Several optimization-based methods have emerged during the past decade, but the one introduced in [CW16] is considered to produce SOTA results in a white-box setting [Che+17a]. In [CW16], the authors test several objective functions and adapt Equation 4.2 to the following formulation for the optimisation problem. Given an objective function $g_{obj} : \mathcal{X} \times \mathcal{Y} \to \mathcal{Y}$ [1], an input $\boldsymbol{X} \in \mathbb{R}^n$, with n the dimension of the vectorized input space and $\boldsymbol{X}^*$ (resp. y') the targeted adversarial example (resp. class):

$$\begin{aligned} &\text{minimize } \|\boldsymbol{X}^* - \boldsymbol{X}\|_2^2 + c \times g_{obj}(\boldsymbol{X}^*, y'), \\ &\text{s.t } \boldsymbol{X}^* \in [0,1]^n \end{aligned} \tag{4.3}$$

The best objective function proposed in [CW16] makes use of the logit vector $\boldsymbol{Z}(\boldsymbol{X}) \in \mathbb{R}^{|\mathcal{Y}-1|}$ (see Definition 1.4) associated with input $\boldsymbol{X}$ and is given by

$$g_{obj}(\boldsymbol{X}, y') = \texttt{max}\left(\left(\underset{i \neq y'}{\texttt{max}} \boldsymbol{Z}(\boldsymbol{X})_i - \boldsymbol{Z}(\boldsymbol{X})_{y'}\right), -k\right), \tag{4.4}$$

with k an hyperparameter set to 0 [Che+17a]. While this method generates strong adversarial examples, *i.e.*, adversarial examples with minimal perturbation on the original image, it requires full knowledge of the target to compute precisely the gradient, which is not always a practical solution. Therefore, alternative methods must be designed for more realistic threat models.

4.2.3 Black-Box Scenario

In a black-box scenario, an attacker has no longer access to any information about the DNN, and the output is restricted to the hard label, which contains far less information. In such settings, no prevailing methodology has emerged. However, several studies, both software- and hardware-based, have demonstrated strong results. One popular software

1. In practice the ouput space can either be the soft-labels or the hard-labels.

method is the *Boundary Attack* introduced in [B18]. This attack starts from a point in the targeted class y' for a targeted scenario, or any different class except the one corresponding to the true class of the input image in an untargeted scenario, and then minimizes the distance to the input image while staying in the region associated with class y'. While intuitively simple, this method achieves very strong results and generates adversarial examples with low perturbation, even compared to gradient-based methods such as [CW16]. Other notable software-based attacks include the *HopSkipJumpAttack* [CJW20] or *QEBA* [Li+20], which efficiently generate adversarial examples by introducing a method to estimate the gradient in black-box settings at the decision boundary of the targeted model.

While in a black-box scenario, the information on the model is limited, it is possible to use physical attacks to increase this knowledge. For example, in [NSF21], the authors observe, using SCA, a correlation between the number of activated neurons in ReLU networks, *i.e.*, neurons whose activation value is superior to zero, and the success of the attack. They use a timing attack to infer the number of activated neurons in the model for a given input, and successfully generate adversarial examples in a black-box scenario. On the other hand, fault injection methods are also used for evasion attacks in an adaptation of adversarial examples. The idea is to perform fault injection to alter the value of a set of weights or directly on the activation values, which causes errors in the internal processing compared to the nominal scenario. These errors propagate within the model and potentially could induce a misclassification. While such attacks do not modify the input directly, they evade the prediction of the models by altering internal values and then by using the high linearity of DL models to propagate these errors, similarly to adversarial examples [GSS14]. Several attacks based on different fault-injection methodologies have been introduced. In [Bre+18], the authors use a laser to induce a fault in the processing of several activation functions and cause a misclassification. Rakin *et al.* [RHF19] target DNN using data encoded in floating-point values following the IEEE format and show that even randomly flipping bits leads to misclassification. This is due to the possibility of flipping the exponent value of the data, which leads to a massive error. For quantized models, this random search is no longer optimal, and several methods considering different threat models have been introduced, *i.e.*, access to the gradient or the soft-labels. An interested reader may refer to the following survey for further information [KHS21].

4.2.4 Gray-Box Scenario and Transfer-Based Attacks

Under our threat model, the vast majority of attacks fall within the gray-box threat model. While a wide range of methodologies has been introduced, this thesis mostly focuses on attacks that do not require any assumption about the model, but assume access to the soft-labels. We can regroup most of the attacks in this category into two main types: *score-based* attacks and *transfer-based* attacks.

Score-based Attacks. In score-based attacks, the attacker uses the soft-labels to guide the generation of adversarial examples using various methods. This includes attacks such as the *One-Pixel Attack* [SVS17], in which the attacker restricts its ability to modify the input image to a subset of the input's pixels. Another popular method is to use the soft-labels to estimate the gradients of the model with respect to the input and then perform optimization similarly to what is done in a white-box scenario. This method was introduced in [Che+17a], in which the authors estimate the gradients using finite differences, showing that it was possible to generate powerful adversarial examples in gray-box settings. This method was then successively extended to improve the gradient estimation [IEM19; Tu+18].

Transfer-based Attacks. On the other hand, transfer-based attacks aim at training a *substitute* model, on which the attacker has full control, to mimic the targeted model. Then, the substitute model is used to generate adversarial examples using white-box attacks, which will be used against the targeted model. This method was first introduced in [Pap+16; PMG16]. Due to its simplicity, several new methodologies based on this principle have emerged [Liu+17; Yan+22; ZYW24]. While the similarity of the substitute model toward the targeted model largely impacts the performance of the attack, adversarial examples have been shown to transfer between different models, [Liu+17; PMG16], and transfer-based attacks pose a serious threat to DNNs.

Adversarial examples are a major threat to the integrity of DNNs. They enable an attacker to significantly reduce the accuracy of a model on specific images and can completely fool a DNN. However, to produce strong adversarial examples efficiently, the threat model needs to be restrictive. One solution coming from transfer-based attacks may be to increase the attacker's knowledge prior to the evasion attack. In the next section, we present one method to increase this knowledge: model extraction attacks.

4.3 Model Extraction Attacks

4.3.1 Principle and Objectives

While the primary purpose of model extraction attacks is to access private information on the targeted model, we subdivide model extraction attacks against DNNs into two groups. The first one targets the hyperparameters of DNNs, *e.g.*, the architecture, the training hyperparameters, and the second directly tries to extract the *functionality* of the model itself. In this thesis, we will only mention attacks that fall into the second category. However, interested readers may find further details on both types of attacks in [OMR23].

Even with this subdivision, several reasons exist for an attacker to extract the functionality of a DNN. The most direct one is to steal the model for a competitive advantage and avoid the complex training and data collection steps. However, as we have seen in the previous section, increasing the attacker's knowledge also enables more powerful attacks. Therefore, model extraction attacks can also be seen as a preliminary step to a more sophisticated attack that requires white-box access, such as data reconstruction.

Jagielski *et al.* [Jag+20] were the first to introduce this duality in the objectives of model extraction attacks. They introduce the distinction between *accuracy-based* model extraction attacks and *fidelity-based* model extraction attacks.

Definition 4.2 (Accuracy-based model extraction [Jag+20]) *Let $\mathcal{D}_{\mathcal{T}}$ be the distribution of the training set and labels $\mathcal{X} \times \mathcal{Y}$, and $f_{\boldsymbol{\theta}}$ the targeted model trained on a dataset from this distribution. Then the goal of an accuracy-based model extraction attack is to extract a model $f_{\hat{\boldsymbol{\theta}}}$ from the targeted model $f_{\boldsymbol{\theta}}$ such that the probability $\Pr_{(\boldsymbol{X},y)\sim\mathcal{D}_{\mathcal{T}}}[\texttt{argmax}(f_{\hat{\boldsymbol{\theta}}}(\boldsymbol{X})) = y]$ is maximized.*

A perfect accuracy-based model extraction attack would therefore produce a model with perfect accuracy on the task for which the targeted model was trained, independently of the performance of the targeted model. On the other hand, fidelity-based model extraction is defined accordingly:

Definition 4.3 (Fidelity-based model extraction [Jag+20]) *Let $\mathcal{D}_{\mathcal{S}}$ be a distribution over $\mathcal{X}$, $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ the targeted model, and $S : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ a similarity function. Then, the purpose of a fidelity-based model extraction attack is to extract a stolen model $f_{\hat{\boldsymbol{\theta}}}$ such that the probability $\Pr_{\boldsymbol{X}\sim\mathcal{D}_{\mathcal{S}}}[S(f_{\boldsymbol{\theta}}(\boldsymbol{X}), f_{\hat{\boldsymbol{\theta}}}(\boldsymbol{X}))]$ is maximized.*

A typical similarity function S, and the one used in [Jag+20], is the label agreement between the targeted model and the stolen one. The limit of fidelity-based model extraction is *functionally equivalent* extraction, which leads to an exact copy of the targeted DNN and is defined by:

Definition 4.4 (Functional equivalence [Jag+20]) *Two models $f_{\boldsymbol{\theta}}$ and $f_{\boldsymbol{\gamma}}$ achieve functional equivalence on $\mathcal{X}$ if for any distribution of input $\mathcal{D}_{\mathcal{X}}$, the following equality holds:*

$$\forall \boldsymbol{X} \in \mathcal{X}, f_{\boldsymbol{\theta}}(\boldsymbol{X}) = f_{\boldsymbol{\gamma}}(\boldsymbol{X}).$$

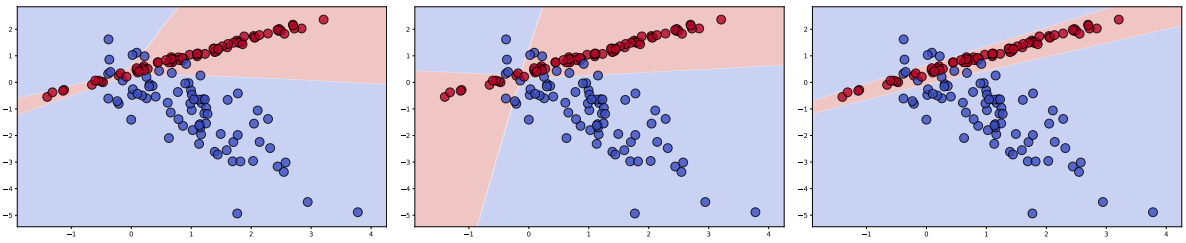
Functional equivalence corresponds to the hardest objective for an attacker [Jag+20], and as such, it is common to use a relaxed definition of functional equivalence:

Definition 4.5 ((ϵ, δ)-functional equivalence [CJM20]) *Two models $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\gamma}}$ are (ϵ, δ)-functionally equivalent on $\mathcal{X}$ if:*

$$\mathbb{Pr}_{X \in \mathcal{X}}[|f_{\boldsymbol{\theta}}(X) - g_{\boldsymbol{\gamma}}(X)| \leq \epsilon] \geq 1 - \delta.$$

Note that Definition 4.5 is designed for regression tasks but can be easily extended to classifiers.

An illustration of the difference between fidelity and accuracy-based model extraction of a classifier is provided in Figure 4.4. We can observe in Figure 4.4 that the model



(a) Classification made by the targeted model.

(b) Classification after a fidelity-based extraction.

(c) Classification after an accuracy-based extraction.

Figure 4.4 – Illustration of the difference between fidelity- and accuracy-based model extraction.

extracted through accuracy-based model extraction has a higher accuracy than the one

obtained via fidelity-based model extraction. However, it is also evident that both models learned very different separations for the task and that fidelity-based model extraction led to a model closer to the original, in terms of decision boundary, than accuracy-based model extraction. This difference in the objective leads to the need for different metrics to evaluate the success of the attack.

Metrics for extraction attacks. Indeed, for accuracy-based model extraction attacks, the main objective is to maximize accuracy on the task, and, therefore, accuracy is used directly as a metric. On the other hand, for fidelity-based model extraction attacks, the idea is to maximize the similarity between the stolen and the targeted model, which can be measured in different ways. The most direct one is to compute the label agreement between the two models. This metric is referred to as *fidelity*. However, due to the complexity of modern DNNs, it is possible for two DNNs to have completely different internal representations and yet produce similar predictions. In such cases, the stolen model might have a strong fidelity, as a metric, with the targeted model, but the level of attacker knowledge on the targeted model has not significantly improved, and more complex attacks such as the ones in Subsection 4.2.2 might fail. Therefore, new metrics are needed. A more fine-grained approach is to compute the absolute maximum error on the stolen parameters and the true ones $\texttt{max}|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}|$. While this metric does not directly compare the labels, it provides a strong characterization of the similarity between the two models. Another approach is to compute the Accuracy under Attack (AuA) of the targeted model.

Definition 4.6 (Accuracy under Attack) *Let $f_{\boldsymbol{\theta}}$ (resp. $f_{\hat{\boldsymbol{\theta}}}$) be the targeted (resp. stolen) model, then the Accuracy under Attack of the targeted model corresponds to the accuracy of the model $f_{\boldsymbol{\theta}}$ on adversarial examples generated on the model $f_{\hat{\boldsymbol{\theta}}}$.*

This metric is used as a proxy to evaluate the impact of fidelity-based model extraction before conducting more powerful attacks.

In this thesis, we mostly focus on fidelity-based model extraction attacks of DNNs. As such, we describe these attacks in Subsection 4.3.3 and Subsection 4.3.4. However, we first present several accuracy-based model extraction attacks for comparison in Subsection 4.3.2.

4.3.2 Accuracy-Based Model Extraction Attack

Accuracy-based model extraction attacks were introduced in [Tra+16]. Most of these attacks on DNNs are based on the principle of supervised learning. An attacker uses the targeted model as an oracle to labelize a dataset and then trains a substitute model on this dataset [OSF18; Bar+20]. In its basic form, this method is close to knowledge distillation (see Subsection 2.2.4). Several factors impact this method, such as the architecture, the dataset used by the attacker, and the type of output of the targeted DNN. Finally, most of these attacks require access to the confidence scores to augment the quantity of information from the targeted model, and as such, are not completely in black-box settings.

While the choice of architecture for the stolen model is important, it is not as crucial as having some knowledge on the distribution of the training data [Tru+20]. In an ideal scenario, an attacker has access to the true distribution of the training data of the targeted DNN. However, this is a strong assumption. Therefore, several attacks have emerged using generated data by a Generative Adversarial Network (GAN) [Goo+14], for example [Bar+20], or through optimization methods [KPQ20].

4.3.3 Fidelity-Based Model Extraction Attack

While supervised learning methods are primarily used for accuracy-based model extraction attacks, they can be adapted to fidelity-based model extraction attacks as in [BM24]. In [BM24], Biton Dor *et al.* use a genetic algorithm to find samples in low-confidence areas of the targeted DNNs to improve the fidelity of the stolen model, resulting in a lower AuA of the targeted model. However, because the supervised learning method is based only on the outputs of the DNN, it is not possible to guarantee the similarity between internal representations and, therefore, between the extracted and the targeted model.

To mitigate this issue, an alternative method based on supervised learning under internal constraints has been introduced in [Rak+21] and extended in [Hec+23]. Both methods first use fault injection, RowHammer in [Rak+21] and laser injection methods in [Hec+23], to extract the Most Significant Bits (MSBs) of the targeted model weights. Then they train their substitute model on a modified loss function (see Subsection 1.2.2), incorporating a penalty term for differences between the MSBs of the stolen and the

targeted model:

$$\mathcal{L}_{MSB}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) = \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) + \sum_{l=1}^{L} ||\boldsymbol{W}^l - \boldsymbol{W}^l_{mean}||, \tag{4.5}$$

with $\mathcal{L}_{MSB}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y})$ the modified loss function, $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y})$ the original loss function, *i.e.*, the cross-entropy loss in [Rak+21], L the depth of the model, $\boldsymbol{W}^l$ the matrix of weights from the substitute model corresponding to layer l and $\boldsymbol{W}^l_{mean}$ a matrix of estimated weights from the extracted MSB. Despite the fact that these methods offer a stronger similarity between the internal representations, they are still based on supervised learning, which is not enough to guarantee functional equivalence (see Definition 4.4).

One possibility to guarantee functional equivalence is to consider side-channel attacks targeting the weights through DEMA (see Subsection 3.2.3) as introduced in [Bat+19] or timing attacks as in [MBC21]. In [MBC21], the authors profile the timing of the multiplication between floating-point values and of the ReLU function to extract either the input or the weights of the models. The idea is that since the timing of the multiplication is known for each result, then if one operand of the multiplication, *i.e.*, the weight or the input vector, is also known, it is then possible to deduce the other one. Maji *et al.* demonstrate the viability of their method on several platforms, including an ARM Cortex M0+, and propose a dedicated countermeasure that will be further described in Subsection 4.4.2. In [Bat+19], Batina *et al.* propose to target the Hamming weight of the accumulated sum between the weight vector of a neuron and its input vector via CEMA on an ARM Cortex-M3 and an ATmega328P.

However, several limitations, such as the impact of the ReLU function, error propagation, and bias extraction, have been brought to light in [Jou+23a]. This method was extended to a GPU in [Hor+25] using a clone device, and several limitations from [Jou+23a], such as the extraction of the bias, have been addressed. However, no results have been provided for the complete extraction so far.

On the other hand, alternative methods based on equations, which do not require physical access, were introduced. The first example of this work was presented in [Tra+16] on logistic regression, and was extended to DNNs using the ReLU function by Maji *et al.* [Mil+18] and Carlini *et al.* in [CJM20]. Due to its similarity to cryptanalysis, this method is now referred to as the *cryptanalytical extraction* of DNNs. In this thesis, we will mostly focus on this type of method. Therefore, we further describe this methodology in the next

section.

4.3.4 Cryptanalytical Extraction of Deep ReLU Neural Networks

Maji *et al.* [Mil+18] were the first to describe how an attacker could extract the weights of a DNN based on the ReLU function. The intuition behind this method is that this function creates a *hyperplane* whose equation is directly connected to the weights of the neuron. This method was implemented and demonstrated on neural networks with one hidden layer by Jagielski *et al.* in [Jag+20], and further extended to deeper architectures in [RK19; CJM20]. Both methods in [RK19; CJM20] extract the DNN's weights by iteratively targeting each neuron, monitoring its state, to find its *critical points*. The critical points and the state of a neuron η can be defined in the following way:

Definition 4.7 (Critical point and state of a neuron [CJM20]) *Given an input $\boldsymbol{X} \in \mathcal{X}$, let $V(\eta; \boldsymbol{X})$ be the function characterizing the input of the neuron η before applying the activation function. Then $\boldsymbol{X}$ is said to be a critical point to the neuron η if $V(\eta; \boldsymbol{X}) = 0$. If $V(\eta; \boldsymbol{X}) > 0$ (resp. $V(\eta; \boldsymbol{X}) < 0$) then η is said to be active (resp. inactive).*

The induced hyperplane of a neuron can be defined accordingly:

Definition 4.8 (Neuron-induced Hyperplane) *The hyperplane of a neuron η corresponds to the set $\mathcal{H}_\eta \subseteq \mathcal{X}$ where $\forall \boldsymbol{X} \in \mathcal{H}_\eta, V(\eta; \boldsymbol{X}) = 0$.*

An example of critical points, neuron states, and a hyperplane is provided in Figure 4.5. By finding enough critical points, it is possible to determine the equation of the neuron-induced hyperplane. This can then be used to find the weights' values with high precision, since by definition, the equation of the hyperplane follows

$$\sum_{i=0}^{n} \boldsymbol{\theta}_{\eta,i} \boldsymbol{X}_i + \beta_\eta = 0, \boldsymbol{X} \in \mathbb{R}^n \tag{4.6}$$

with $\boldsymbol{\theta_\eta}$ (resp. β_η) the weight (resp. the bias) vector associated with the neuron η. It is important to notice that for a neuron η with n parameters, the induced hyperplane is an $(n-1)$-dimensional piecewise-linear surface [HR19]. This means that the weight vector can only be extracted up to a scaling factor. In practice, we set the first weight $\boldsymbol{\theta}_{\eta,0}$ of $\boldsymbol{\theta_\eta}$

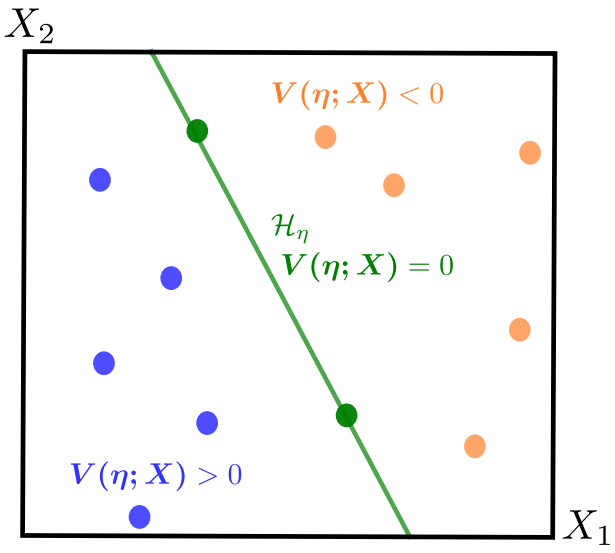


Figure 4.5 – Neuron-induced hyperplane.

as the scaling factor and from Equation 4.6 we obtain:

$$\boldsymbol{X}_0 = \sum_{i=1}^{n} \frac{-\boldsymbol{\theta}_{\eta,i}}{\boldsymbol{\theta}_{\eta,0}} X_i - \frac{\beta_\eta}{\boldsymbol{\theta}_{\eta,0}}, \text{ , s.t. } \boldsymbol{\theta}_{\eta,0} \neq 0 \tag{4.7}$$

After the extraction of a critical point, it is possible to retrieve the neuron's weights up to a *scaling factor* by sampling along the standard basis vector of $\mathcal{X}$. This process is defined in [CJM20] as signature search. The neuron's signature can be defined in the following manner:

Definition 4.9 (Neuron signature [Can+24]) *The signature of a neuron η corresponds to a vector $\boldsymbol{S}_{\boldsymbol{\eta}} \in \mathbb{R}^n$, for which there exists a scalar $\alpha \in \mathbb{R}$ such that $\boldsymbol{\theta}_{\boldsymbol{\eta}} = \alpha \cdot \boldsymbol{S}_{\boldsymbol{\eta}}$.*

In the previous example, the vector $\boldsymbol{S}_{\boldsymbol{\eta}} = (1, \frac{\boldsymbol{\theta}_{\eta,1}}{\boldsymbol{\theta}_{\eta,0}}, \ldots, \frac{\boldsymbol{\theta}_{\eta,n}}{\boldsymbol{\theta}_{\eta,0}})$ is a signature of the neuron η and $\boldsymbol{\theta}_{\eta,0}$ is the scaling factor.

Since the ReLU function is equivariant under positive multiplication, *i.e.*, $\forall x \in \mathbb{R}$ and $\forall c \in \mathbb{R}^+$, $ReLU(c \cdot x) = c \cdot ReLU(x)$, extracting the signature of the neuron instead of its true vector of weights is not an issue. Indeed, if the sign of the input of the ReLU function is respected, then the activation value of each neuron can later be re-scaled during the extraction of the neurons' signature in the following layers. This means that the value of the scaling factor α of the neuron's signature is not important, only its sign. Therefore, this method cannot extract the true targeted model but rather an equivalent neural network,

which, in case of a successful attack, is functionally equivalent to the targeted model. This is due to the existence of transformations, such as the positive scaling of the weights or the permutation of a neuron within a layer, as presented in [Mar+23], that are indistinguishable by an attacker just by monitoring the output of a model [RK19; Jag+20], making exact extraction of a ReLU-based DNN impossible in these settings [Jag+20].

Under these conditions, the model extraction attack can be seen as an iterative process over the neurons based on two steps:

1. Extraction of the neuron's signature.
2. Extraction of the sign of the scaling factor to ensure that the neuron's state remains the same.

Extraction of the neuron's signature. By considering the targeted model $f_{\boldsymbol{\theta}}$ as an oracle, the search for a neuron's signature can be performed directly using its input-output pairs. Indeed, if the targeted model is composed of piecewise linear activation functions, then $f_{\boldsymbol{\theta}}$ is also piecewise linear and discontinuities can be observed in the gradient of $f_{\boldsymbol{\theta}}$. These discontinuities correspond to each change of state by a neuron in the DNN. More specifically, the discontinuity in the gradient of the ReLU function is localized at 0, which implies that by monitoring the gradient of the targeted model, critical points for each neuron can be found. However, performing the signature's search using these discontinuities is still challenging and requires high assumptions. Firstly, the target model's output needs to be either the confidence scores or the logits, to allow the attacker to estimate the gradient through finite differences. Secondly, since it is impossible to determine to which neuron each critical point corresponds, it is necessary to sample a large number of critical points to ensure that each neuron will have at least one. Thirdly, in deeper layers, on average, half of the neurons are deactivated [CJM20], which implies that for a given critical point, a large number of its entries are null, and as such, the associated weights cannot be recovered. Carlini *et al.* [CJM20] solve this issue by recombining signatures from different critical points. However, one must first ensure that the signatures correspond to the same neurons in the same layers. To mitigate this, Carlini *et al.* use a consistency check under the assumption that signatures related to different neurons or layers will lead to inconsistent results. Finally, the output needs to be returned in a high-precision format to get the most precise estimation. In [Jag+20; CJM20] the authors extracted ReLU-based networks running double-precision, *i.e.* 64-bit floating-point data.

Extraction of the neuron's sign. While neuron signatures can be revealed through the discontinuities of the gradient, the sign of the scaling factor is not accessible directly by an attacker. Carlini *et al.* [CJM20] solve this by testing all of the 2^m possibilities, with m the number of neurons in the layer. As this becomes rapidly impractical for large DNNs, Canales-Martínez *et al.* [Can+24] propose another method based on activating specific neurons through crafted inputs to extract the sign of each neuron. Their method successfully extracts the sign of each neuron within a model with 8 hidden layers, each with 256 neurons, for which the signatures were previously extracted.

This process is repeated for each neuron within each layer sequentially, starting from the first layer to the last one. This attack is therefore a gray-box attack, as it assumes knowledge of the architecture and access to the model's confidence scores. However, in the past few years, alternative methods have been introduced to enable the extraction in a more restrictive threat model, and more specifically by removing access to the soft-labels.

Extraction in hard-label settings. Chen *et al.* [Che+25a] were the first to propose a method for cryptanalytical extraction of DNNs designed for binary classification in the hard-label settings. Their method is based on *decision boundary points*, *i.e.*, points on the transition boundary between the two labels, and was demonstrated on DNNs with two hidden layers. However, their method has an exponential time complexity [Car+25] and is restricted to binary classifiers, which is impractical, leading researchers to consider alternative methods. Such a method was introduced by Carlini *et al.* in [Car+25]. In this work, the authors also use the decision boundary points [2] between the classes, but in this case, to detect the critical points. Indeed, due to the piecewise linearity of the ReLU function, the input space of the DNN is split into *cells* in which the DNN behaves as a linear function. In previous methods, the boundaries between two cells, which correspond to the neuron-induced hyperplane (see Definition 4.8) were detected using the gradient, which is no longer possible in this threat model. The intuition of the attack presented in [Car+25] is that the decision boundary, which is visible in hard-label settings, must be a flat hyperplane within a cell boundary. Therefore, it is possible to detect what they refer to as dual points, *i.e.*, points both on the decision and the cell boundary, by monitoring the shape of the decision boundary. An illustration of these points on an MLP with one hidden layer of five neurons taking two-dimensional vectors as input is provided in Figure 4.6.

2. The authors in [Car+25] refer to these points as transition points. Out of clarity we choose to use the notation decision point in this thesis.

Since by definition these points are also critical, it is then possible to extract a signature (see Definition 4.9) using the dual points.

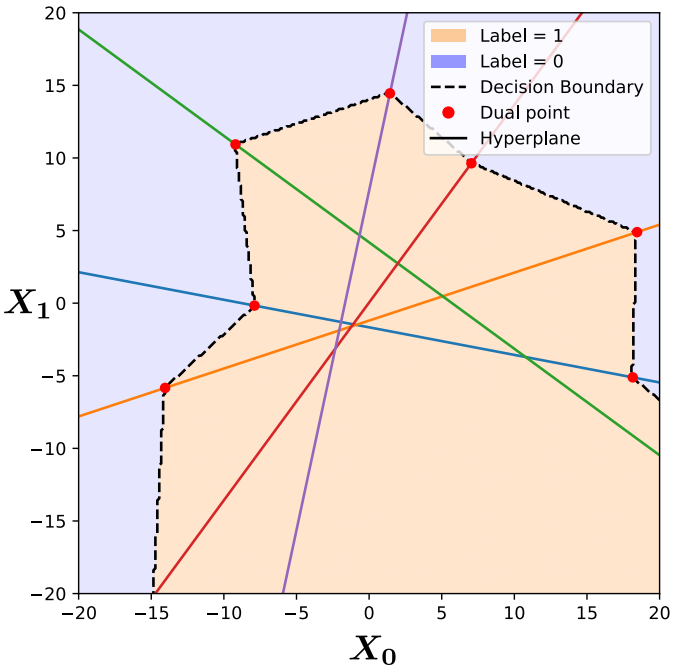


Figure 4.6 – Illustration of the concept of dual points as introduced in [Car+25] on a MLP with one hidden layer of five neurons each associated with a different hyperplane represented by the colored lines.

Extraction of the last layer. Due to the absence of the ReLU function in the last layer, the extraction process differs from the one for the previous layers. In soft-label settings, the attacker can use the confidence scores as labels for supervised learning [Jag+20]. However, in hard-label settings, the labels do not contain enough information to ensure functional equivalence, and the extraction of the last layer has been left out in [Car+25]. Recently, a first methodology has been proposed in [CS25]. Similar to [Car+25], they use points on the decision boundary to form a system of equations and extract the parameters from the last layer. Indeed, such points, by definition, correspond to points for which two classes have an equal probability of being selected, and a system of equations can be created. The issue is that it is not possible to obtain a full-rank system for all models on all tasks. The authors of [CS25] solved this problem by noticing that it was possible to fix a certain number of variables in the system of equations to obtain an equivalent network. While

this method does not enable the exact extraction of the targeted model, the extracted model displays perfect fidelity toward the targeted model [CS25].

Model extraction and evasion attacks represent a significant threat to developers trying to ensure the confidentiality and integrity of their deployed models. Therefore, several countermeasures have been proposed by the community to detect or mitigate these attacks. In the next section, we will describe these countermeasures, as well as some natural difficulties that emerge from the constraints related to embedded systems.

4.4 Countermeasures and Difficulties Related to Embedded Systems

4.4.1 Countermeasures for Adversarial Examples

Adversarial examples are inherent to DNNs [GSS14], and therefore, it is always possible to craft an input that evades the decision of the classifier. The goal of countermeasures against adversarial attacks is to make the creation of such examples as difficult as possible, either by making the generation of adversarial examples costly in terms of resources and detectable or by degrading the quality of the adversarial example, *i.e.*, increasing the necessary perturbation to a level noticeable by a human observer. More generally, the countermeasures can be classified into *proactive* methods, which actively modify the targeted model to make it more robust to adversarial examples, and *reactive* methods, which aim to detect or remove adversarial examples. Reactive methods tend to be specifically designed for a type of adversarial example and mostly operate under the assumption that the attacker does not know which defense is used, making them less popular than proactive defenses [Ber21]. Therefore, we will only introduce proactive methods in this section.

One of the most intuitive defenses is to restrict the attacker's access to the gradient, so that the optimization of Equation 4.2 becomes more difficult. Countermeasures based on this idea are referred to as *obfuscated-gradient* methods [ACW18]. While this method has been widely applied [Guo+18; Xie+18], Athalye *et al.* demonstrate in [ACW18] that this method does not provide security, by completely circumventing six out of the nine defenses based on this method from a test set composed of non-certified defenses claiming white-box robustness from ICLR 2018. A popular alternative is to use adversarial training [GSS14; Mad+18], which consists of adding adversarial examples to the training set. This

method is not impacted by the remark from [ACW18] as it does not obfuscate the gradient. However, it does not scale well for large datasets such as ImageNet [KGB17]. Finally, one promising defense has been introduced in [Léc+18] and then extended in [CRK19]. In this work, the authors transformed their model into a smooth version and obtained guarantees on the output of the model for an input within an l_2 ball. The idea is that small adversarial perturbations are contained within the radius of this ball, and therefore, the model is unaffected.

4.4.2 Countermeasures for Model Extraction Attacks

As mentioned in Section 4.3, several methodologies, including software- and hardware-based approaches, have emerged targeting the confidentiality of the model. Therefore, both software and hardware-based countermeasures have been introduced to mitigate these attacks. In [CDG20], the authors use CNNs to approximate a noisy version of the identity function. The idea is that the hyperplanes induced by the neurons of the CNNs make the extraction of the weights following the method from [RK19; CJM20] both more difficult and noisier.

In [MBC21; Bat+19; Tak+22], the timing difference of the activation function is used to gain information on either the architecture of the model, the input, or the parameters themselves. Therefore, a constant implementation of the ReLU function has been proposed in [MBC21] to mitigate this threat. However, this does not reduce the impact of differential attacks and, therefore, masking of the model has been proposed in [DCA19; DCA20]. While these works show promising results, they are not currently implemented in traditional frameworks, and even more, vulnerable implementations to timing attacks can still be found, as in [Ma20].

4.4.3 Natural Difficulties in Embedded Systems

Additional practical limitations should be considered in the embedded context, as hardware constraints might force the neural network to use a lower-precision format for weights, activations, and input-output pairs. This makes gradient estimations noisier, which has an impact on optimisation-based methods both for evasion and model extraction attacks. While the logits or confidence scores might be given alongside the prediction in an MLaaS context, for embedded systems, it is frequent that the output is restricted to only the class with the highest probability. For example, FINN [Umu+16] is a pop-

ular framework for deploying low-precision neural networks on FPGAs, and by default, only the *top-1* prediction is returned as output. Finally, as mentioned in Subsection 2.2.5, parallelized architectures such as FPGAs or TPUs are now commonly used for deploying DNNs. This leads to parallelized operations, which implies that the corresponding signal will be more difficult to analyse from a side-channel analysis point of view. As such, attacks against these accelerators are rarer and often require a first step of reverse-engineering to ease the leakage identification [Hor+25; Gon+24].

While hardware attacks have been used to perform evasion and model extraction attacks, these often correspond to new methodologies that are disjoint from traditional software attacks. We argue that hardware attacks can be used in conjunction with software attacks to relax the threat model and even extend the potential of such attacks. In the next part of this thesis, we will propose two new attacks based on this principle. In Chapter 5, we will describe how side-channel attacks can be used as a first step for evasion attacks to move from a gray-box to a black-box threat model. And then in Chapter 6, we will introduce a new divide-and-conquer approach based on cryptanalytical extraction and a side-channel attack to extend the potential of fidelity-based model extraction attacks.

Part II

Combining Software and Hardware Attacks to Enhance Evasion and Model Extraction Attacks

Chapter 5

Improving Evasion Attacks Through Side-Channel Profiling of the softmax Function

In this chapter, we investigate how SCA can reduce the assumptions necessary for attacks against DL algorithms. More specifically, we describe a methodology that enables an attacker requiring access to the logits in an evasion attack to move from a gray-box access to a complete black-box scenario by first performing a side-channel attack. First, we recall the limitations associated with traditional score-based attacks, and then present our methodology. Finally, we evaluate our attack both as a standalone methodology to extract the logits and as part of a global methodology combined with the ZOO attack [Che+17a] to generate adversarial examples in black-box settings. The work proposed in this chapter has been presented at AISec 2023 [Coq+23] and also at ESSAI [Coq+24b] as an invited talk.

Contents

5.1 Motivations 81
5.1.1 Limitations on Score-Based Attacks 81
5.1.2 Side-channel Analysis as a First Block in Black-Box Attacks . . 82
5.2 Overview of the Attack 84
5.2.1 Global methodology 84
5.2.2 General Observations 86
5.3 Practical Experiments 88
5.3.1 Threat Model 88
5.3.2 Experimental Setup 89
5.3.3 Analysis of the Hardware Vulnerability 90
5.4 Evaluation 93

5.4.1 Logit Extraction 93
5.4.2 Complete Attack 95
5.5 Conclusion 97

5.1 Motivations

5.1.1 Limitations on Score-Based Attacks

While evasion attacks have been proposed in black-box settings [CJW20; B18] and achieve strong results in terms of induced perturbations to create an adversarial example, these typically require far more queries to craft an adversarial example compared to gray box attacks [B18]. Indeed, in a gray-box scenario, an attacker can use several supplementary pieces of information to perform the attack, such as prior knowledge of the architecture or knowledge of the training distribution. However, we argue that one of the most critical pieces of information an attacker can have in a gray-box scenario compared to a black-box one is access to the soft-labels.

Access to the soft-labels enables an attacker to obtain far more information on the model compared to the strict hard label. It can be used to estimate the gradient as in [IEM19; Che+17a; Tu+18], which is a key step in the generation of adversarial examples. While this chapter focuses on evasion attacks, the soft-labels can also be used to find low-confidence regions to train substitute networks [BM24] and can even be used to extract directly the weights in ReLU-based DNNs [Jag+20; RK19; CJM20].

As mentioned, score-based attacks such as [IEM19; Che+17a; Tu+18] improve performance in terms of queries compared to black-box attacks, even though the only assumption present in their threat model that characterizes them as gray-box is access to the soft-label. Therefore, such attacks could be easily considered in a black-box scenario if a previous black-box attack could extract the soft-labels. Since the main advantage is the query efficiency, and the soft-labels are used in an iterative process as part of an optimization procedure, this attack needs to be able to extract the soft-labels efficiently. Consequently, in this chapter, we propose to introduce a preliminary side-channel attack to effectively retrieve the soft-labels at each step of the optimization process.

5.1.2 Side-channel Analysis as a First Block in Black-Box Attacks

In [Che+17a], the authors introduce their methodology as a supplementary block on top of the Carlini and Wagner attack [CW16]. The idea is to use an efficient and strong white-box evasion attack and remove some assumptions to make the attack more practical. Therefore, they adapt the loss function from [CW16] Equation 4.4 using the soft-labels instead of the logits directly. In their work, the objective function to minimize is defined as:

$$g_{obj}(\boldsymbol{X}, y') = \texttt{max}\left(\left(\max_{i \neq y'} \texttt{log}(f(\boldsymbol{X})_i) - \texttt{log}(f(\boldsymbol{X})_{y'})\right), -k\right), \tag{5.1}$$

with k a hyperparameter set to 0. Finally, they use finite differences as a proxy for the gradients obtained via backpropagation in a white-box setting. Given a scalar $h \in \mathbb{R}$ and a unit vector $\boldsymbol{E} \in \mathbb{R}^n$:

$$\frac{\partial f}{\partial \boldsymbol{X}_i} \approx \frac{f(\boldsymbol{X} + h\boldsymbol{E}_i) - f(\boldsymbol{X} - h\boldsymbol{E}_i)}{2h} \tag{5.2}$$

In this chapter, we propose to exploit side-channel observables of embedded DNNs to perform template attacks and DLSCAs (see Subsection 3.2.3) to extract the value of the logits vector by targeting the operations during the computation of the softmax layer. Since this layer is used to transform the logits independently from the previous ones into probabilities, our attack is free from any restrictive assumptions on the architecture. Our attack can therefore be seen as a supplementary block on top of the work proposed by [Che+17a] as represented in Figure 5.1.

In terms of the threat model, our attack does not increase the level of knowledge the attacker has from an AI perspective. Indeed, the only assumption made is the presence of the softmax function, which is commonly accepted as a standard for the majority of multiclass classifiers. However, as mentioned in the previous section, the idea is to maintain query efficiency during the generation of adversarial examples. Therefore, we choose a profiled side-channel attack so that the number of traces, *i.e.*, of queries, during the creation of adversarial examples remains as low as possible to keep the advantage of score-based attacks compared to other black-box methodologies. In consequence, our side-channel attack requires access to an open device for the profiling phase to create the templates. This

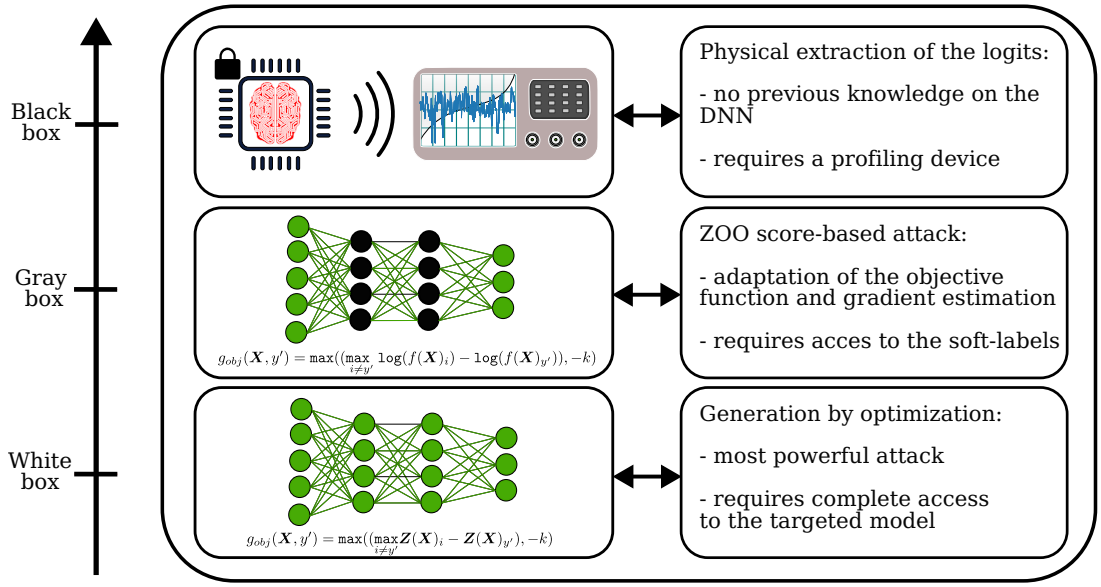


Figure 5.1 – Visualisation of the different blocks of our attack and their corresponding threat model.

implies that our overall methodology corresponds to a profiled black-box attack. [1]

Although this chapter focuses on the generation of adversarial examples using the ZOO methodology, we would like to emphasize that the side-channel attack presented here is specific neither to this task (evasion attack) nor to this methodology (ZOO). Since we choose to extract the logits, all attacks exploiting such values could use our side-channel attack as a first step. For example, the objective function in [Che+17a] could be modified to the original function from [CW16], which utilizes the logits. Furthermore, in [Ues+18], the authors directly exploit the logits to approximate the gradient also via finite differences. As such, our side-channel attack could be integrated directly within their work.

Finally, as previously mentioned, both model extraction methods proposed in [RK19] and in [Jag+20] need access to the soft-labels to efficiently steal the model's parameters. With our methodology, an attacker could first conduct side-channel attacks to extract the logits in a black-box setting before applying the attacks described in [RK19] and [Jag+20]. In other words, the main purpose of our methodology is to demonstrate the suitability of

1. As mentioned in Chapter 4, in this thesis, the threat model, *i.e.*, white-box or black-box, is related to the attacker's knowledge of the DNN, not to the type of access to the physical system.

hardware attacks in making software attacks more practicable in a black-box context by introducing a methodology to remove a previous assumption, *i.e.*, the access to the logits or soft-labels.

5.2 Overview of the Attack

5.2.1 Global methodology

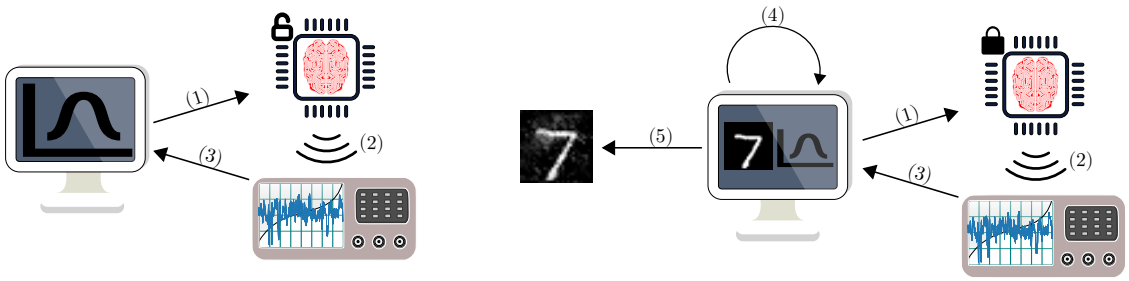
As previously mentioned, the requirements of ZOO [Che+17a] and SPSA [Ues+18] are not fulfilled in black-box settings due to the supposed access to the soft-labels or the logit vector. In this context, our methodology takes advantage of hardware attacks to generate SOTA adversarial examples by providing access to the logits and the probability vector in a black-box setting. This forms an overall assumption-free attack, *e.g.*, no prior knowledge of the DNN's hyperparameters or parameters is required, nor access to the logits, confidence scores, or even output labels.

The first step of our attack is to construct a distinguisher function that precisely maps the side-channel traces to the targeted variables. This is a crucial step that needs access to the copy, D_O, of the targeted device D_T. As explained in Definition 3.7, D_O enables the malicious user to create a statistical distinguisher, which will be used to attack the softmax function of the targeted device D_T running the neural network. This step corresponds to the profiling phase performed on D_O. The process, illustrated in Figure 5.2a, can be detailed as follows:

- **Step (1)** A set of random input data, for which the logits are known, is sent to the embedded DNN for prediction.
- **Step (2)** During prediction, the attacker captures the related side-channel traces, which contain information on the manipulated logits.
- **Step (3)** The attacker creates a distinguisher function, *e.g.*, template, DLSCA, to characterize the leakage model related to the manipulation of the logits.

These steps require a strong assumption that we will detail in Subsection 5.3.1.

Once these steps are performed, our complete attack methodology, *i.e.*, side-channel attacks and generation of adversarial examples, can be illustrated as in Figure 5.2b. It follows the same general steps as a classical ZOO or SPSA attack by querying the network with an input and progressively transforming this input until an adversarial example is



(a) Profiling phase on D_O.

(b) Attack phase on D_T.

Figure 5.2 – Schematic description of the profiling and attack phases of our attack.

created. This is an iterative process with one iteration being composed of the following steps:

- **Step (1)** One data value is sent n times to the embedded neural network for prediction.
- **Step (2)** Meanwhile, the attacker captures the n corresponding side-channel traces containing information related to the manipulated logits.
- **Step (3)** The attacker performs the logit extraction attack using the constructed distinguisher function, *e.g.*, templates or DNN models, and the newly captured attack traces. Then, the extracted logits are provided to the adversarial example methods, *e.g.*, ZOO or SPSA, in our case.
- **Step (4)** Using a gradient-free method, *e.g.*, ZOO, SPSA, the attacker estimates the perturbation to add to the inputs to create a successful adversarial example that will fool the embedded neural network.

Since a single iteration is usually not sufficient to construct a successful adversarial example, steps (1) to (4) are repeated until the perturbation causes a misclassification from the network, *i.e.*, step (5) in Figure 5.2b, or until the generation framework stops. Therefore, if we denote m_{adv} the maximum number of iterations allowed to create an adversarial example and n the maximum number of traces necessary to successfully extract the logits at each step (1-3), the global complexity C of the attack can be written as:

$$C = \mathcal{O}(n \cdot m_{\text{adv}}). \tag{5.3}$$

In comparison with traditional score-based approaches, the overall attack complexity is multiplied by the number of attack traces required to retrieve the logits added to the number of traces necessary for the profiling phase. Although the number of queries for

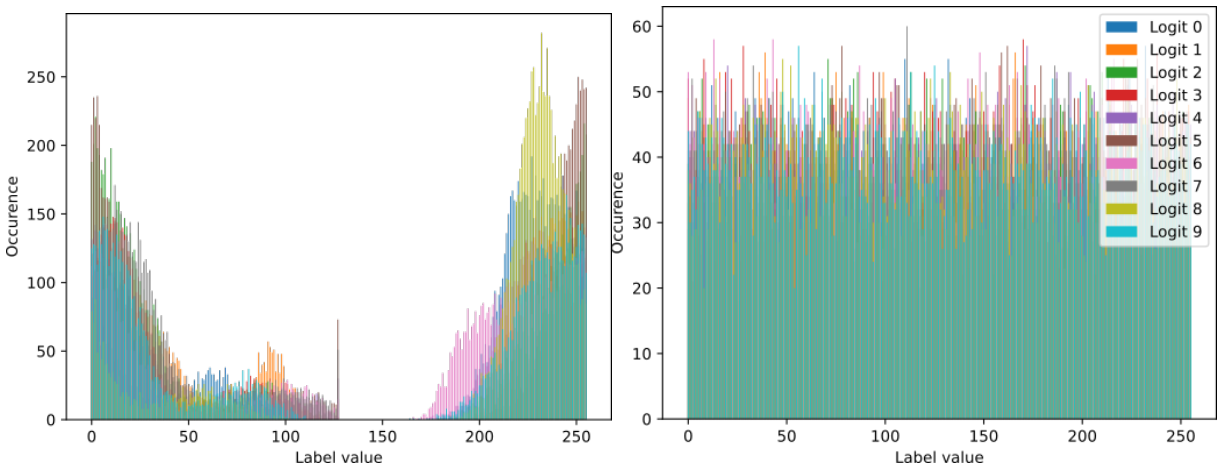
the profiling phase may be large, once this phase is completed, an attacker can target any DNN using a softmax function on the same type of targeted device D_T in black-box settings. However, as shown in Equation 5.3, the practicability of the attack scenario highly depends on the ability of the profiled side-channel attack to extract the logits in a small number of attack traces.

5.2.2 General Observations

Choice of target for logit extraction. As mentioned in the previous section, the main idea in our attack consists of retrieving the logits manipulated by the targeted embedded neural network through the use of side-channel attacks. In particular, the targeted operations induced in a layer or an activation function should probably leak information on the logits. Since the softmax function is computed to approximate the output probability distribution $\texttt{Pr}[\boldsymbol{Y}|\boldsymbol{X}]$ from the logits for a given input $\boldsymbol{X}$, it is a natural target from a side-channel perspective for extracting information related to these sensitive variables. Furthermore, this function is the most common in multiclass classifiers and is used as a final transformation independent of the previous ones. Therefore, attacking the softmax function makes our methodology independent of the architecture and applicable to all neural network architectures that solve a classification task and apply this function. Finally, while side-channel attacks have been implemented against DNNs, to the best of our knowledge, no previous side-channel attack has targeted the softmax function. As such, no countermeasures are implemented to protect the device against these threats.

A natural distribution difficulty. As stated in the previous sections, template attacks and DLSCAs can be considered to extract the corresponding logits in a profiled scenario. However, for an unbiased side-channel attack, the profiling phase must be conducted with the logits uniformly distributed over $\mathcal{L}$ (see Definition 1.4). The under-representation of certain logits will lead to poor estimation of the distribution parameters, *i.e.*, the pair $(\boldsymbol{\mu}_l, \Sigma_l)$ for template attacks, and the weights for DLSCAs, leading to a deterioration of the attack performance. This issue is especially true when targeting the logits of a DNN. Indeed, the purpose of the embedded neural network is to select an output among the possible classes in $\mathcal{Y}$. As such, the DNN is trained to maximize one value in the logits vector and minimize the others as much as possible. It is then likely that the distribution of the logits will be over-represented for the lower values of the output range and under-represented in the higher part. An example of class distributions is provided in Figure 5.3a.

The logits, in this example, are represented using 8-bit signed integers, and values greater than 127 correspond to negative logits. We can then observe that the distribution follows the pattern of over- and under-representation of classes that we previously described.



(a) Non-uniform distribution.

(b) Uniform distribution.

Figure 5.3 – Example of the distribution of logits on a real use-case over a set of $10,000$ samples.

If the adversary targets an open-source DL framework, *e.g.*, PyTorch [Pas+19], Tensorflow [Aba+16], FINN [Umu+16], NNOM [Ma20], the source code can be modified to force the uniform distribution of the logits during the profiling phase. Because most of the embedded neural networks are generated from open-source DL frameworks, the attacker can naturally have access to the source code. Then, the attacker can modify this code on the open device D_O to mitigate this issue, as illustrated in Figure 5.3b. Given the fact that our attack is architecture-independent, this enables the malicious user to perform a balanced profiling phase and construct an unbiased side-channel attack applicable to all targeted embedded devices D_T using the same open-source DL framework as in D_O. We argue that this method, although requiring modification of the source code, does not compromise the black-box assumption. Indeed, another method for a malicious user is simply to increase the number of acquisition traces and use a subset to balance the dataset. Furthermore, other attack methods less impacted by this distribution, such as Stochastic attacks [SLP05], could be considered. However, we consider them out of the scope of this chapter and leave the improvement of this attack using such methods for future work.

In our work, we choose to modify the implementation on the open device, and the following section proposes to experimentally validate this new attack strategy.

5.3 Practical Experiments

5.3.1 Threat Model

In this section, we provide context for the threat model we consider in our attack. Following Section 5.2, the primary goal of the attackers is to extract the logits from the embedded neural network by performing a side-channel attack. As mentioned, our strategy is based on a profiling attack scenario, namely template attacks and DLSCA. Such a scenario suggests that the attacker has physical access to a copy of the targeted device. Therefore, we consider attackers who obtain the same hardware reference D_O as the targeted device D_T on the market. As it is assumed that the DL framework used to embed the DNN model on the targeted device is known, the attacker can build its own model on D_O to extract the logits from D_T. Indeed, as the attack scenario introduced in Section 5.2 is architecture-independent, *i.e.*, the softmax function is targeted, the knowledge of the DNN structure embedded on D_T is not required. Consequently, the attacker can easily use any architecture on D_O as long as it includes the same targeted function embedded as in D_T, *i.e.*, the softmax function, to satisfy the requirements of the profiling attack scenario. Once this profiling phase has been performed on D_O, the attacker can extract the logits from the targeted device D_T during the inference stage. In other words, once the targeted embedded neural network has already been trained and deployed. For example, we consider attackers who come from the DNN accelerator design team or are insiders in the companies hosting the DNN infrastructure, allowing them to gain physical access to the device.

Attacker capability. To conduct such a scenario, it is assumed that the attacker knows the DL framework embedded into the targeted system. Knowing the reference of the targeted hardware device, the adversary can modify the source code of the DL framework to mitigate the issues introduced in Subsection 5.2.2. Secondly, attackers can acquire high-resolution ElectroMagnetic (EM) traces of the targeted embedded neural network during the inference phase by using an oscilloscope. These assumptions align with the scenario previously introduced. From a DL point of view, as the attack strategy targets a function of the DL framework that is independent of the underlying neural network architecture, *e.g.*, the softmax function, no prerequisite on the DNN is needed except knowledge of the task that is being performed, *i.e.*, the number of classes.

5.3.2 Experimental Setup

Hardware implementation. We validate our contribution by targeting an ARM Cortex-M7 MCU on an STM32-F767 board running at a frequency of 216 MHz. This board incorporates 2 Mbytes of flash memory and 512 Kbytes of SRAM, which is enough to load quantized models on the board. We choose to target a classifier based on the DenseNet [HLW16] architecture, including a softmax as the last layer. The model is trained on the MNIST dataset [Den12] with the Adam optimizer [KB15] and is developed with Tensorflow [Aba+16]. The MNIST dataset is composed of $60,000$ training images and $10,000$ test images corresponding to images of handwritten digits and has been used as a benchmark for image classification tasks. Since this dataset considers 10 classes, 10 logits must be extracted to validate the method introduced in Section 5.2. In the context of embedded systems, we choose to use quantization to reduce the impact on the resources. The inputs, weights, and activation values of the model are quantized to 8 bits, using deterministic PTQ.

NNOM, an open-source framework, is used to facilitate the incorporation on the MCU [Ma20]. NNOM translates the model from a TensorFlow object to a C project, which can then be directly built and run on the board. The generated project is compiled without optimization by the compiler. We decide to test only on the NNOM framework, but we expect our attack method to translate to other frameworks, as any unprotected manipulation of the logits in the assembly code will induce side-channel leakages. We leave the evaluation of other frameworks and hardware implementations for future work.

Practical setup. A probe from Langer (EMV-Technik RF-U 2,5 with a frequency range going from 30MHz up to 3GHz) is connected to an amplifier (ZLF-2000G+) to capture the EM side-channel traces. To acquire the side-channel traces, a Lecroy oscilloscope (2.5 GHz WaveRunner 625Zi) is used in our setup. To ease the attack exploitation, a trigger surrounding the targeted function, *i.e.*, `local_softmax_q7()` from NNOM, has been added. As the softmax computation is distinguishable from the other patterns, the configuration of the trigger is not considered difficult. A quick overview of our experimental setup is illustrated in Figure 5.4.

To gain a better insight into our method, we conduct 10 independent attacks, providing a success rate metric to identify the number of attack traces required to successfully retrieve each logit. As a reminder, the success rate metric defines the probability that an attack succeeds in recovering the true logit amongst all hypotheses in $\mathcal{L}$.

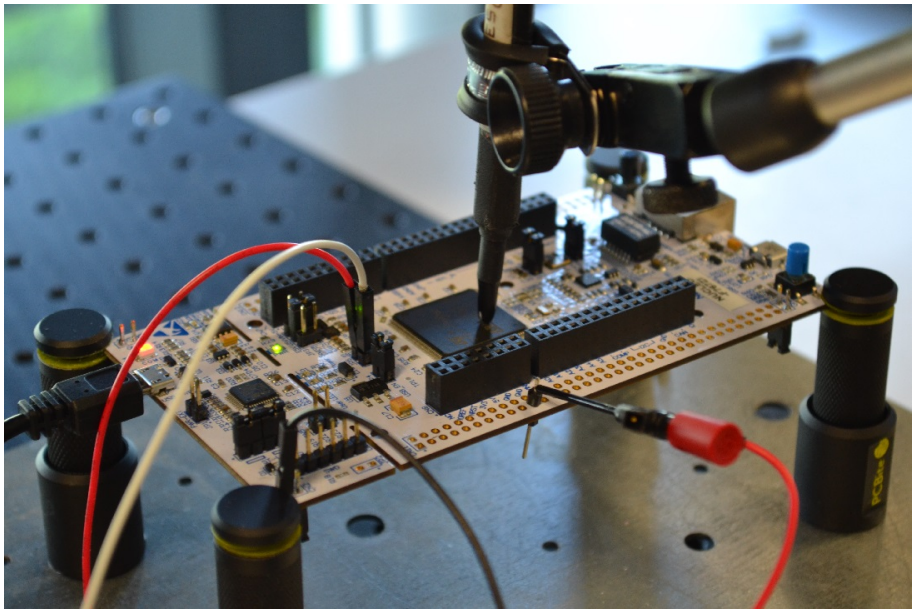


Figure 5.4 – Experimental setup composed of an EM probe placed on the chip of the MCU with the trigger connected to an external pin.

Logits distribution. Following the recommendations provided in Subsection 5.2.2, we evaluate the distribution of the logits on the trained embedded neural network. As expected, a non-uniform distribution is observed (see Figure 5.3a). To solve this issue, we modify the source code of the NNOM softmax function to force its input, *i.e.*, the logits, to follow a uniform distribution. The results we obtained are highlighted in Figure 5.3b. This requirement is highly recommended to mitigate the bias in the distinguisher function, *i.e.*, template attacks, DLSCAs, but is not mandatory, as stated in Subsection 5.2.2.

5.3.3 Analysis of the Hardware Vulnerability

To validate the methodology introduced in Subsection 5.2.1, the NNOM softmax function is targeted. We choose this implementation as it is a common solution in the literature [ZR20; Rus+23; Jou+23a; Hec+23]. Whereas the traditional softmax function is defined by a normalization that maps the logits to a probability distribution following $s(\boldsymbol{Z}(\boldsymbol{X})_i) = \frac{e^{\boldsymbol{Z}(\boldsymbol{X})_i}}{\sum_{j=1}^{|\mathcal{Y}|} e^{\boldsymbol{Z}(\boldsymbol{X})_j}}$, where $\boldsymbol{Z}(\boldsymbol{X})_i$ denotes the logits related to the ith class for the input $\boldsymbol{X}$, the NNOM framework proposes an alternative to this natural approach. In particular, instead of computing the exponentiation of each logit as suggested in the classical softmax function, NNOM captures the maximum value $\texttt{max}_z$ from the vector of logits $\boldsymbol{Z}$. Then, all logits lower than $(\texttt{max}_z - 8)$ are ignored, while others are normalized following

Listing (5.1) Part of the `local_softmax_q7` implementation in NNOM.

```
1  /* Base is initialized to (
     int16_t)-257 */
2    /* We first search for the
     maximum */
3    for (i = 0; i < dim_vec; i
     ++)
4    {
5        if (vec_in[i] > base)
6        {
7            base = vec_in[i];
8        }
9    }
```

Listing (5.2) Assembly code of line 5 from Listing 5.1.

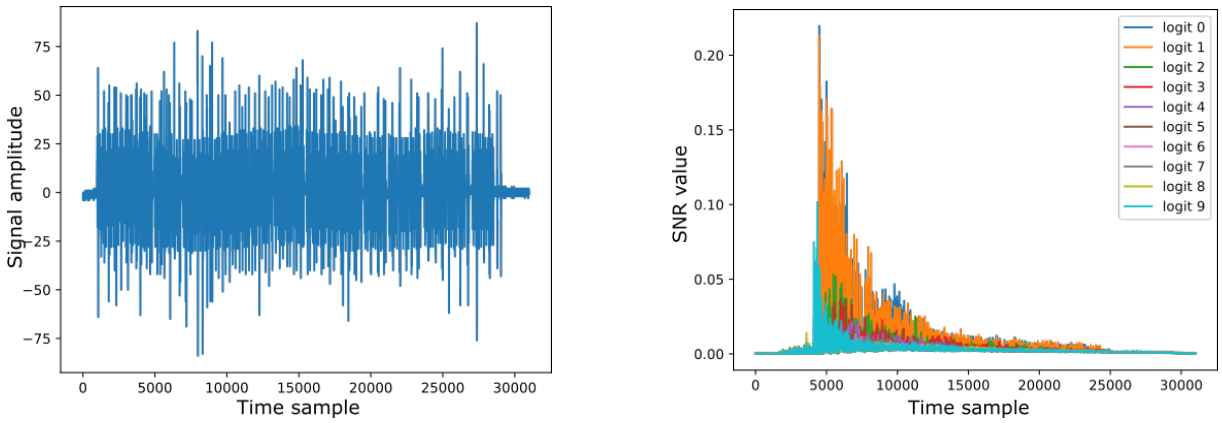
```
1 ; if (vec_in[i] > base)
2 ldr r3, [r7, #32] ; Loading the
     address of the index "i"
3 ldr r2, [r7, #12] ; Loading the
     address of z
4 add r3, r2        ; Set the
     pointer to the i-th element
     of confidence score z
5 ldrsb.w r3, [r3]  ; Loading of
     the i-th element of z
6 sxth  r3, r3      ; Extension the
     i-th element of z to a 32-bit
7 ldrsh.w r2, [r7, #30] ; Base
     value loading
8 cmp r2, r3        ; Comparison
     between base and the i-th
     element of z
```

Figure 5.5 – C and Assembly codes from the NNOM softmax function.

the function `local_softmax_q7`(). In Listing 5.1, we provide an extract of the listing associated with the C code source of the softmax computation from NNOM. This portion searches for the maximum value from the `vec_in` vector, which characterizes the vector of logits $\boldsymbol{Z}(\boldsymbol{X})$.

For each element included in `vec_in`, the base value is updated if its value is lower than the ith element of `vec_in`, *i.e.*, $\boldsymbol{Z}(\boldsymbol{X})_i$. Listing 5.2 defines the assembly code which characterizes the "if"-condition involved in line 5 of Listing 5.1. The logit $\boldsymbol{Z}(\boldsymbol{X})_i$ stored in the Flash memory is loaded into register `r3` (see line 5 in Listing 5.2). During the load instruction, a side-channel attack can be performed to extract information related to the targeted logit. In addition, the `base` value is updated if the "if"-condition is respected. Therefore, when the `base` value is loaded into register `r2` (line 7 in Listing 5.2), leakages could be observed on the current maximum value in `vec_in`. This load instruction suggests the manipulation of some value $\boldsymbol{Z}(\boldsymbol{X})_i$ multiple times, which can introduce a dependency between the targeted variable $\boldsymbol{Z}(\boldsymbol{X})_i$ and the side-channel traces.

We want to confirm our analysis of this security flaw by selecting a temporal window in which the signal is directly associated with the value of logits. Figure 5.6a represents one EM trace associated with the processing of the softmax function for different logits during our acquisition window. Figure 5.6b displays the SNR (see Equation 3.2) for the



(a) EM trace of the softmax execution.

(b) SNR associated with each logit.

Figure 5.6 – Preprocessing of an EM trace.

Table 5.1 – Preprocessing on the EM traces depending on the obtained SNR value.

	Logit 0	Logit 1	Logit 2	Logit 3	Logit 4	Logit 5	Logit 6	Logit 7	Logit 8	Logit 9
Maximum SNR value	$2.3 \cdot 10^{-1}$	$2.2 \cdot 10^{-1}$	$5.0 \cdot 10^{-2}$	$3.5 \cdot 10^{-2}$	$2.0 \cdot 10^{-2}$	$2.0 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$1.6 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$
SNR threshold	$5.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$
Dimension of reduced traces	6,514	8,840	1,654	1,302	774	2,866	2,223	5,813	6,186	6,299

first logit computed. As expected from our analysis of Listing 5.1, the ratio between the leakage associated with this variable and the noise changes during the execution. This displays the manipulation of the targeted variable in the assembly code. We would like to highlight that the scenario proposed in this chapter can be conducted on any type of DL implementation since the manipulation of the logits occurs in the assembly code.

To validate our attack method, for each logit, a set of 1,900,000 EM traces, each comprising 31,000 time samples, is captured for the profiling set, while 1,000 traces are acquired for the attack phase. To reduce the need for computation, the EM traces are reduced by selecting a subset of PoIs (see Definition 3.5) based on the SNR peaks we obtain on each logit. For this purpose, a threshold is configured for each logit such that all time samples associated with an SNR peak exceeding this threshold are kept. The threshold is defined to find the best trade-off between computational time and leakage information. The results we obtained are defined in Table 5.1. While we use a thresholding method based on the SNR, other dimensionality reduction techniques, such as PCA [Arc+06; SA08; CDP16], LDA [CK15; Bru+15], or KDA [CDP17], could be suitable alternatives.

5.4 Evaluation

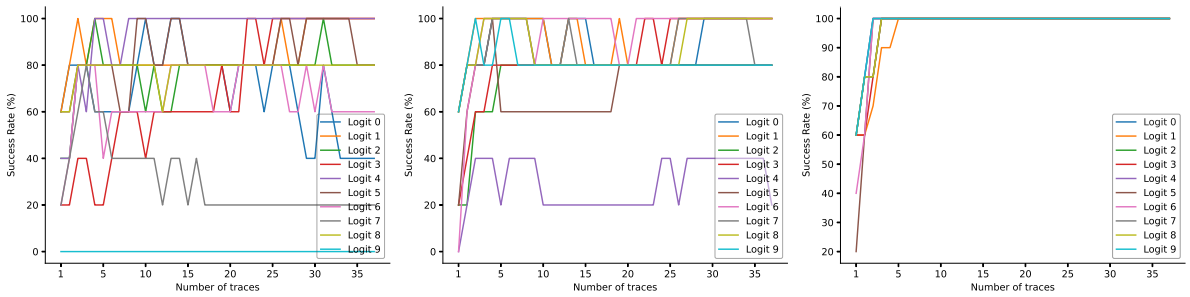
5.4.1 Logit Extraction

To evaluate the logit extraction process, three attacks have been conducted. All attacks have the same number of profiling traces on the open device D_O. On the contrary, the number of attack traces used on the targeted device D_T to achieve a similar average success rate is different from one attack to another. This metric will be used to compare the efficiency of the three methods.

First, template attacks [CRR03] are performed to assess the ability of the Gaussian distribution to capture the dependencies between the targeted logits and the EM traces. However, this assumption cannot hold in practice, so we evaluate a more generic solution, which considers a broad family of models: Multinomial Logistic Regression (MLR). As suggested by Masure *et al.* [Mas+23], this class of models defines the set of all polynomial transformations such that a clear connection can be proposed with template attacks. Finally, to gain a comprehensive understanding of the method, MLPs are constructed to perform a DLSCA and recover the logits of the embedded neural network. While the latter solution is beneficial from a performance perspective, the lack of interpretability makes its configuration a difficult task. In the following paragraphs, we present the results obtained with each method.

Template attack. The template attacks are the naive solution when profiled attacks are conducted. The obtained results are displayed in Figure 5.7a, where each colour of a curve corresponds to the success rate for a given logit (out of 10) as a function of the number of traces used for the attack phase on D_T. For example, the purple curve corresponds to the success rate for the 5th logit out of the 10 possible classes, averaged on 10 independent experiments. For this specific logit, 8 attack traces are required to successfully extract the targeted logit value, *i.e.*, achieve a success rate of 100%.

From Figure 5.7a, it can also be observed that after only 11 attack traces, the average attack success rate is around 66%. In addition, while some logits are recovered with a very small amount of traces, *e.g.*, 8 traces for the 5th logit, others are not successfully retrieved, *e.g.*, no successful attack is obtained when the 10th logit is targeted. This observation suggests that further investigations can be conducted to improve the performance of template attacks. An attacker could increase the number of collected traces on the open device D_O to improve the profiling phase and build a more effective solution distinguisher



(a) Template attacks. (b) MLR attacks. (c) DLSCA attacks.

Figure 5.7 – Success rate results on targeting NNOM softmax function. Each colour of a curve corresponds to the success rate of a given logit (out of 10) as a function of the number of traces used for the attack.

function. Other choices for attack or setup configurations, such as dimensionality reduction techniques, different SNR thresholds, preprocessing, and probe positions, could also significantly improve the performance of our attack. The results we obtain illustrate the practical limitations of the template attacks. To mitigate this issue, an attacker can use another technique based on MLR to consider a less restrictive class of models.

Multinomial Logistic Regression. To investigate the benefits of this class of models, a simple MLP without hidden layers or activation functions is configured. Only an output softmax function is used to solve the multiclass classification problem. During the profiling phase, we use the traces collected on D_O and the associated labels, *i.e.*, the logits, to train the MLP via supervised learning using the Adam optimizer [KB15]. This profiling process is conducted over 20 epochs with a batch size (resp. the learning rate) set to 512 (resp. 10^{-5}).

In comparison with the template attack result, Figure 5.7b shows a performance improvement when the MLR is considered. In particular, after 10 attack traces, the average success rate is about 80%, and with this method, all the logits are extracted at least once, which was not the case with the template attack. However, the attack on the 5th logit performs poorly in comparison with other logits. Consequently, if an attacker wants to follow the methodology introduced in Subsection 5.2.1, the extraction of the logits must be perfect, as a wrong estimation will impact the generation of adversarial examples. Even if some SOTA results suggest that the knowledge of the top-k logits is sufficient to generate adversarial examples [Ily+18; IEM19], this scenario is not considered in the scope of this work, and therefore, we must improve the extraction.

Deep-Learning based Side-Channel Attacks. Finally, the most generic solution consists of designing a DNN model to approximate the best possible distinguisher function. We target the embedded system developed to solve the classification problem on the MNIST dataset running on the targeted device D_T, with another network trained on the EM traces of the logit's processing on the open device D_O. While the latter is trained to solve the classification problem on the MNIST dataset, the former is designed to exploit the EM traces and extract the logits included in the embedded neural network. The MLP trained on the traces of D_O is composed of 3 hidden layers of $1,000$, $1,000$, and 100 neurons, respectively. To approximate non-linear functions, each neuron uses the ReLU activation function. The same configuration as for the MLR is used, *i.e.*, 20 epochs and a batch size of 512, whereas the learning rate is adapted to 10^{-4}. The results obtained with this setting are included in Figure 5.7c. Interestingly, when the DL technique is used to extract the logits from the embedded neural network, all the sensitive variables are retrieved within 5 EM traces. This method offers the best performance compared to the others, and as such, this is one that we adopt in our global methodology (see Subsection 5.2.1).

Through this process, it can be confirmed that an attacker can perform side-channel attacks to extract the logits from the embedded neural network. Regarding the global complexity of the attack (C in Equation 5.3), the attacker needs $5 \times m_{\text{adv}}$ attack traces to create a given adversarial example. In our experiment, we set m_{adv}, the maximum number of iterations to create an adversarial example, to $10,000$. Consequently, if the generation of adversarial examples is possible, the total number of attack traces required to conduct an end-to-end attack is $50,000$ in the worst case [2]. However, it should be mentioned that, sometimes, $10,000$ iterations are not sufficient to generate such adversarial examples using the ZOO framework. Consequently, the total number of attack traces highly depends on the performance of the adversarial example tool. The following section investigates an end-to-end scenario of our proposition using the ZOO framework.

5.4.2 Complete Attack

Once the extraction is performed using DLSCAs, an attacker can generate adversarial examples in a black-box setting using gradient-free frameworks. To validate this statement,

2. In practice, to defeat a protected cryptographic system, a side-channel attack may require millions of traces.

Table 5.2 – Comparison between our methodology and generation on a substitute network.

	Attack rate (%)	Average Distortion (L2)	Number of queries
Logits extraction and ZOO	100	7.54	$1.9 \cdot 10^6$
Substitute network	56.61	5.80	6000

we use the open-source implementation of the ZOO attack [3] provided by [Che+17a] to attack our embedded model. Since this framework uses the probability vector in the objective function, we compute it with the extracted logits and solve the optimization problem accordingly. We confront the results of our attack with those obtained after white-box generation using the Basic Iterative Method [KGB18] from the Cleverhans framework [Pap+18] on a substitute network. The substitute network is based on the same architecture as the targeted network, and we use the same methodology as [Rak+21], training it using 10% of the training data labeled by the embedded model. We evaluate the attack rate, *i.e.*, percentage of adversarial examples generated that are misclassified by the targeted DNN, the average distortion, *i.e.*, the $L2$ distance between the original and the adversarial example, and the average number of queries to generate an adversarial example. Our results are included in Table 5.2.

The number of queries necessary to produce an adversarial example is much higher when using our methodology. This is caused by the profiling attack, which needs queries to the system to acquire the side-channel traces and build the model. The number of traces for this phase is highly dependent on the implementation and the surrounding noise. Since we are using optimization methods, *i.e.*, DLSCAs, in our profiling phase, this number is highly experimental and could be much lower for another hardware. However, the main advantage of our methodology is its genericity. Once the profiling phase is finished, an attacker will not have to fully retrain it while targeting any models implemented using NNOM. On the other hand, to train the substitute network, the malicious user will need access to training data specific to each task and will potentially have to retrain a substitute for each new targeted DNN. As expected, even in an optimal case with a substitute based on the same architecture, not all white-box-generated adversarial examples are transferable, with an average rate of 56%. Since ZOO directly uses the targeted network, the generated examples are, by definition, directly adversarial for this network. The high value in the distortion for both methodologies seems to be due to the 8-bit quantization of the input, which forces the perturbation to be larger to have an impact.

3. `https://github.com/IBM/ZOO-Attack`

With this design, we demonstrate on a simple example, *i.e.*, the ZOO framework, that an attacker can extend existing SOTA frameworks to generate powerful adversarial examples in a black-box scenario, leveraging our end-to-end methodology. Our method can therefore be adapted to any score-based methods, such as [Tu+18; IEM19], and provide an alternative to previous black-box methodologies, such as [CJW20; Li+20].

5.5 Conclusion

This chapter aims to demonstrate that it is worth considering SCA and, more generally, physical attacks against DL implementation, in addition to software attacks. To do so, we show the ability of an attacker to generate powerful adversarial examples by breaking the black-box assumption through the exploitation of side-channel attacks against an embedded DNN. In particular, we contribute to the SOTA by adding a side-channel extraction of the logits into score-based adversarial example frameworks using these values.

To validate our methodology, a practical scenario has been conducted on a microcontroller implementing a DenseNet with the NNOM framework. Based on a code analysis, security flaws have been identified and exploited in the NNOM softmax function, and the logit vector has been extracted, in the best case, within 5 attack traces. We finalize our attack by combining our physical attack with the ZOO framework for adversarial example generation, confirming the complementarity between software and hardware attacks.

To further improve our contribution, some future works can be suggested. Vulnerability in the softmax function can be studied in some widely used open-source DL frameworks, *e.g.*, PyTorch, Tensorflow, FINN, or on more optimized hardware implementations, *e.g.*, on FPGAs or TPUs. Further investigations on the attack can focus on the impact of incorrect logit estimation on the final attack and on adapting it to perform non-profiled side-channel attacks, which reduce the attacker's capabilities. Finally, while the security of embedded AI systems is still a recent research direction, finding countermeasures against our methodology is crucial to make AI solutions more secure.

In this chapter, we focused on evasion attacks. Nevertheless, as previously mentioned, other types of attacks, such as model extraction, could benefit from a first attack that extracts the logits. Indeed, recently Carlini *et al.* [Car+25] proposed a new methodology to improve cryptanalytical extraction of DNNs (see Subsection 4.3.4) in hard-label settings,

showing the potential benefits of applying the work presented in this chapter to these attacks. However, while their methodology relaxes the threat model, the basic methodology still relies on numerous assumptions. Therefore, we propose to apply the same principle that we use in this chapter, *i.e.*, combining software and hardware attacks, to propose a new methodology. More specifically, we propose to target a different activation function to enhance the cryptanalytical extraction of DNNs.

Chapter 6

Side-Channel Extraction of Deep-ReLU Networks with 32-bit Data

From side-channel attacks to equation-based attacks, several methodologies have been introduced to perform fidelity-based model extraction. As mentioned in Subsection 4.3.3, one of the promising ones is cryptanalytical extraction (see Subsection 4.3.4). By definition, this methodology requires access to the confidence scores, which is one of the principal limitations with the restriction to MLPs. While alternative methodologies operating in hard-label settings have been suggested [Che+25a; Car+25], these are still based on the same principle and therefore have only been demonstrated on MLPs. To target more complex architectures while not returning to threat models requiring access to the confidence scores, we propose an alternative methodology based on a divide-and-conquer approach in this chapter. We first present the current limitations of the SOTA in fidelity-based model extraction. Then, we describe how, using a side-channel attack, it is possible to split a DNN into subnetworks, enabling an attacker to target non-fully connected layers in hard-label settings. We validate our methodology by successfully extracting all the architectures previously targeted in the literature using a simulated implementation of the side-channel attack. We make our code available at `https://anonymous.4open.science/r/Side_channel_cryptanalytic_extraction_of_DNN-3E2C/README.md`. Finally, we target several new architectures implemented on an MCU, such as a truncated MobileNetv1, which was not previously possible.

Contents

6.1 Current Limitations of Fidelity-Based model Extractions in Hard-Label Settings 100

6.1.1 Complexity of the Attack and Special Cases of Neurons 100

6.1.2 Extracting Deep Layers . . . 102
6.1.3 Restriction to Fully-Connected-Layers . . . 103
6.1.4 Combining Cryptanalysis and Side-Channel Attacks . . . 104
6.2 Subdividing Deep Neural Networks to Improve Cryptanalytical Extraction . . . 106
6.2.1 Splitting DNNs at Each Activation Function . . . 106
6.2.2 Improved Signature Extraction . . . 109
6.2.3 Estimation of the Sign of the Scaling Factor . . . 113
6.2.4 Choice of oracle and complete methodology . . . 115
6.3 Practical Experiments . . . 119
6.3.1 Threat Model . . . 119
6.3.2 Experimental Setup . . . 120
6.3.3 Side-Channel Analysis of the ReLU Implementation . . . 122
6.3.4 Complete Extraction of Classifiers . . . 125
6.3.5 Simulation of our Methodology in Soft-Label Setting . . . 127
6.4 Discussion and Limitations . . . 129
6.4.1 Propagation of Errors . . . 129
6.4.2 Impact of the Special-Case Neurons . . . 130
6.4.3 Extraction of the Last Layer . . . 131
6.5 Conclusion . . . 132

6.1 Current Limitations of Fidelity-Based model Extractions in Hard-Label Settings

6.1.1 Complexity of the Attack and Special Cases of Neurons

The most direct limitations of the methodology presented in [Jag+20; CJM20] are the restrictions in the associated threat model:

- **Access to the soft-labels in high precision (64 bits).** Since this methodology is based on estimating the gradient to detect the critical points, access to the soft-labels in a high-precision format is required for precise estimation. In practice, they demonstrate their methodologies on DNNs using 64-bit floating-point arithmetic.

- **Restriction to MLPs.** To create the system of equations, the attacker also requires the architecture of the targeted DNN. Furthermore, the proposed methodology has only been demonstrated on MLPs, and, to the best of our knowledge, no previous work has successfully demonstrated extraction on DNNs that include non-fully connected layers such as max or average pooling layers.

Furthermore, even if an ideal scenario is assumed for the attacker, *i.e.*, all the mentioned assumptions are met, cryptanalytic extraction using the methodology introduced in Subsection 4.3.4 is still challenging for several reasons. A first issue reported in [Can+24] is the special cases of neurons that almost never change state. Two types of special case neurons were described in [Can+24]:

- ***Always-Off.*** These neurons correspond to *dead* neurons. Indeed, their parameters are such that for the majority of the input space, their pre-activation value is negative. Since all previous works on cryptanalytical extraction are demonstrated on ReLU-based DNNs, the output of these neurons is always null, hence the name.
- ***Always-On.*** These neurons correspond to the opposite of the *always-off* neurons, *i.e.*, their pre-activation value is always positive. Therefore, they correspond to linear transformations in ReLU-based DNNs.

For these neurons, the number of queries needed for the search of critical points can exceed the mean number of required queries for a classical neuron by a factor of 10 (see Subsection 6.4.2). Even if these special case neurons are highly difficult to treat in practice, no investigations are provided in the SOTA to deal with such an issue.

Another issue is that the information in the input-output pairs does not allow identification of the neuron for which the critical point is found. Therefore, an attacker cannot validate if all neurons have critical points. Carlini *et al.* [CJM20] solve this issue by sampling a large number of critical points with respect to the number of neurons. This statistically ensures that each neuron has enough critical points to infer the hyperplane required to find the corresponding signature, but it also leads to the search for unnecessary critical points. This might render the attack too complex for deeper architectures due to the large number of required queries.

Finally, the absence of direct mapping of the critical points to the targeted neuron leads to another problem brought to light in [Liu+25], which we will further describe in the following section.

6.1.2 Extracting Deep Layers

While several limitations on the signature extraction and the sign extraction (see Subsection 4.3.4) are directly mentioned in [CJM20], successive works mostly aimed at improving the sign extraction [Can+24; Foe+24] under the assumption that the signature extraction was already made. However, Liu *et al.* [Liu+25] recently exposed a new concern in the signature extraction process introduced in [CJM20]. More specifically, their concern is that the consistency check used by Carlini *et al.* in [CJM20] to merge the incomplete signature of a given neuron (see Subsection 4.3.4) is not correct.

As mentioned in Subsection 4.3.4, in deeper layers, the extracted signatures (see Definition 4.9) are often incomplete due to the ReLU function in the previous layer deactivating some of the components of the input vector. However, to extract the neuron, a complete signature is required. The solution proposed in [CJM20] involves merging signatures to obtain all the entries. To merge signatures, Carlini *et al.* check if shared entries of the partial signature are proportional. If the check is positive, the two signatures can be merged, and the size of the aggregated signature is increased by one, *i.e.*, the size of the aggregated signature corresponds to the number of signatures merged. In [CJM20], the authors argue that an aggregated signature with a size greater than two is on the targeted layer, as it is unlikely that signatures from deeper layers could be merged.

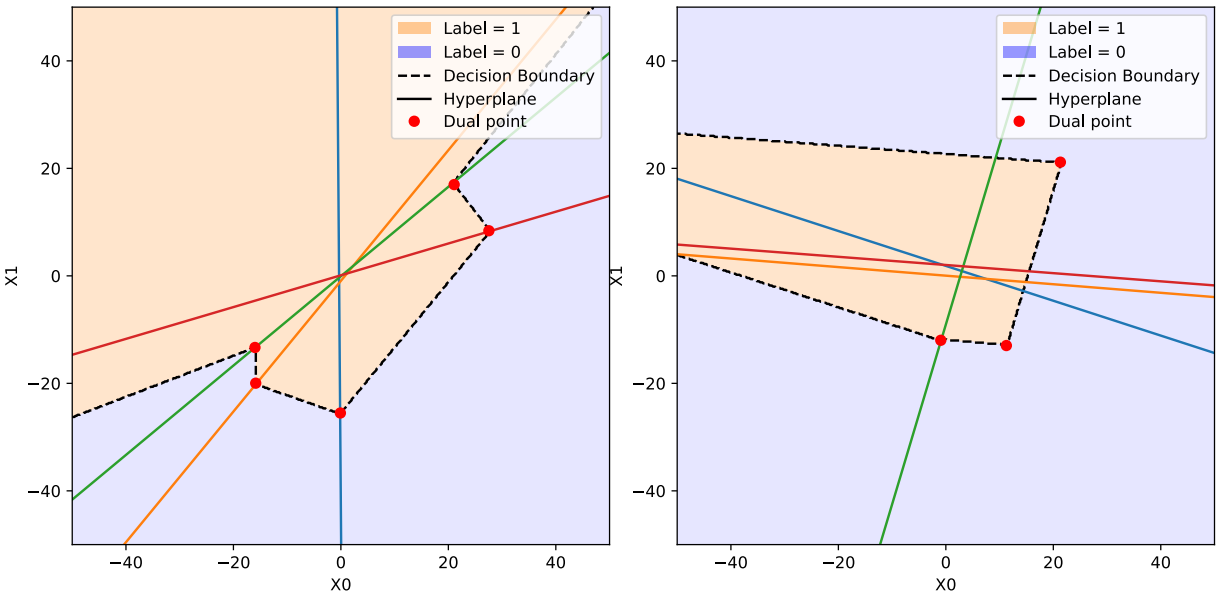
However, Liu *et al.* [Liu+25] demonstrate that when extracting a layer l, two signatures corresponding to neurons in a deeper layer $l+t$ can, in fact, be merged provided that they share the same activation patterns, *i.e.*, the same state for the neurons, across all layers between the targeted layer l and their true layer $l+t$. In this case, the signature can be merged into an aggregated signature with a size greater than two, and the consistency check from [CJM20] fails, leading to incorrect signature recovery. Several solutions to mitigate this issue have been introduced in [Liu+25], which have led to the successful extraction of signatures from deeper layers. More particularly, they are able to increase the SOTA by extracting 95% of the weights in each layer of DNNs with eight hidden layers on MNIST, while previous methodologies could not perform the extraction beyond the third layer [Liu+25]. However, even with their method, targeting very deep architectures remains difficult. Furthermore, while these improvements make the attacks more practical, they do not change the threat model and still consider full access to the soft-labels. As such, several works in the literature have tried in parallel to move from soft-label settings to hard-label settings [Che+25a; Car+25] as described in Subsection 4.3.4. However, while these methods relax the threat model, they still face several limitations.

6.1.3 Restriction to Fully-Connected-Layers

As mentioned in Subsection 4.3.4, the most promising methodology in hard-label settings has been introduced in [Car+25]. In this work, Carlini *et al.* propose using the decision boundary to find dual points, *i.e.*, points that are both critical and on the decision boundary, to extract the trained parameters, *i.e.*, the weights and biases. In their work, they successfully extract all the hidden layers of a four-layer DNN trained on CIFAR-10 using only the hard-labels provided by the targeted model. However, they only demonstrate their methodology on fully connected networks. We argue that their methodology is not directly applicable to more complex architectures, especially those including non-fully connected layers, such as max-pooling layers.

To illustrate this limitation, we train a simple linear neural network with one hidden layer on a binary classification task. In Figure 6.1a, we can see that the shape of the decision boundary only changes when passing through a neuron-induced hyperplane, which is the main assumption of their method. Therefore, in this case, these points effectively correspond to dual points, and the hyperplane can be recovered in hard-label settings as stated in [Car+25]. However, when we train this network on the same task after adding a max pooling layer, we obtain the same level of performance, but we observe in Figure 6.1b that the decision boundary is no longer correlated with the hyperplanes.

This leads to incorrect estimation of the dual points and, consequently, to a wrong estimation of the weights. Therefore, in such a configuration, the approach proposed in [Car+25] is not applicable, and the cryptanalytical model extraction attack fails. This can be explained by the following observation: a change of linear cell may occur, but due to the pooling layer, if the corresponding activation is not large enough to pass through the pooling layer, this change is not visible in the decision boundary at the hyperplane intersection, but once the corresponding activation is larger than the others. From the previous sections, we can see that, despite providing strong results on MLPs in soft-label settings, several limitations still exist, first within the methodology itself as described in Subsection 6.1.2, and most importantly when adapting the methodology to a more realistic threat model. Therefore, an alternative methodology to perform fidelity-based model extraction must be found.



(a) Fully connected network.

(b) DNN with a max pooling layer.

Figure 6.1 – Visualisation of the impact of a max pooling layer on the correlation between the changes in the transition boundary and its intersection with the hyperplanes.

6.1.4 Combining Cryptanalysis and Side-Channel Attacks

In [CJM20], Carlini *et al.* make the comparison between DL models and cryptosystems, and the analogy between the secret key and the weights of the DNN. Therefore, side-channel attacks correspond to a natural solution to fidelity-based model extraction, and it is for this purpose that they are introduced in [Bat+19]. However, as mentioned in Subsection 4.3.3, several challenges related to the targeted device remain to perform a complete extraction in black-box settings. Furthermore, we argue that despite the original concerns on this methodology, DEMA attacks targeting the weights through the accumulated sum become much more complex on parallelized architectures. Indeed, in AI accelerators such as GPUs or TPUs, the matrix product can be optimized so that each accumulated sums are computed in parallel. This parallelization implies that the leakages associated with these values are temporarily intricated, and as such, the exploitation is more difficult. In practice, a first step of reverse engineering is necessary [Hor+25; Gon+24]. Therefore, it is necessary to introduce an alternative methodology that is not based on extracting accumulated sums.

A natural alternative is to consider cryptanalytical extraction, as this methodology is purely software-based and, as such, completely agnostic of the targeted device. However,

as mentioned in the previous sections, these attacks are also based on a large number of assumptions and are impacted by numerous restrictions. Our objective is therefore to propose an alternative methodology for performing fidelity-based model extraction robust to the special case of neurons and able to extract deep networks composed of non-fully connected layers in hard-label settings. Nevertheless, we argue that most of the introduced limitations are caused by the fact that the model is targeted as a whole and are not present in one-layer-deep neural networks, as we will show in Section 6.2.

Consequently, we propose an alternative divide-and-conquer strategy that utilizes a side-channel attack to overcome these limitations. More specifically, our methodology differs from traditional cryptanalytical extraction in the following points:

- We perform the extraction not on the whole targeted DNN but on a subset of smaller DNNs, which, when regrouped, form the targeted DNN. We argue that by splitting a DNN at each activation function, we enable the extraction of more complex DNNs. More precisely, we take advantage of the order in CNNs, *i.e.*, parametric layers, activation function, and then pooling layers as in AlexNet [Kri10] or a group of convolutional layers followed by an activation function, then a pooling layer, and fully connected layers as in [How+17], to bypass the limitations associated with pooling layers.
- Our signature extraction (see Subsection 4.3.4) is performed on each of the smaller DNNs sequentially, and as such, it is not impacted by the soft or hard-label settings or by the same previous restrictions introduced in [Liu+25] regarding the depth of the targeted model.
- The extraction of the sign of the scaling factor is made through an exhaustive search as in [CJM20], but we use our divide-and-conquer approach to reduce the number of hypotheses to one for each smaller DNN.

With this alternative methodology, our goal is to introduce a new attack that is less impacted by the type of device compared to other side-channel attacks, with a broader range of targets compared to current cryptanalysis-based methods.

6.2 Subdividing Deep Neural Networks to Improve Cryptanalytical Extraction

6.2.1 Splitting DNNs at Each Activation Function

In our methodology, we suppose access to an oracle $\mathsf{O} : \mathbb{N} \times \mathbb{N} \times \mathcal{X} \times \mathcal{X} \to \mathcal{B}$, which corresponds to a function returning a boolean value indicating if two inputs $\boldsymbol{X}^1 \in \mathcal{X}$ and $\boldsymbol{X}^2 \in \mathcal{X}$ are associated with the same state for two neurons $(\eta, \gamma) \in \mathbb{N}^2$, *i.e.*, if $V(\eta; \boldsymbol{X}^1) = V(\gamma; \boldsymbol{X}^2)$ (see Definition 4.7). Note that this oracle does not need to know the true states $V(\eta; \boldsymbol{X}^1)$ and $V(\gamma; \boldsymbol{X}^2)$ associated with inputs $\boldsymbol{X}^1$ and $\boldsymbol{X}^2$, only if those two correspond to the same state for the neuron η and γ. As such, the oracle O returns a boolean value, with True corresponding to the case where the two inputs correspond to the same state and False otherwise. Since this function will be often applied to different inputs for the same neuron η, we will directly use the following notation when there is no confusion: $\mathsf{O}(\eta, \boldsymbol{X}^1, \boldsymbol{X}^2) = \mathsf{O}(\eta, \eta, \boldsymbol{X}^1, \boldsymbol{X}^2)$. Moreover, in the rest of this chapter, we will consider that $\mathcal{X} \subseteq \mathbb{R}^n$, with $n \in \mathbb{N}$ the input dimension of the targeted DNN. This oracle naturally removes any need for soft-labels in the search for critical points. Indeed, starting from a random point, an attacker can search for critical points by sampling random points until finding one corresponding to the opposite state and then perform a binary search using the oracle O. A high-level description of our method is presented in Algorithm 2.

Since we perform a binary search, finding a true critical point for the targeted neuron η, *i.e.*, a point $\boldsymbol{X}$ such that $\boldsymbol{X}$ is localized on the hyperplane $\mathcal{H}_\eta$ (see Definition 4.8), *i.e.*, $\boldsymbol{X} \in \mathcal{H}_\eta$, can be challenging due to numerical approximations. Instead, we use a hyperparameter Δ to place a critical point within a determined distance from the hyperplane $\mathcal{H}_\eta$. We use the oracle to compare the states of the neuron for the two inputs at each step of our binary search, and determine which point to update to get closer to the hyperplane. Under the hypothesis that the oracle performs the comparison perfectly, our algorithm is equivalent to a dichotomic search on a sorted list composed of binary values, corresponding to the two possible states of all points on the line between the two initial points $\boldsymbol{X}$ and $\boldsymbol{Y}$, such that $V(\eta; \boldsymbol{X}) \neq V(\eta; \boldsymbol{Y})$. Since we are also using binary search to retrieve the critical points, the complexity of this phase is the same as the one in [Jag+20] and will require $|\mathtt{log}_2(\Delta)|$ queries to place a point within a distance of Δ to the hyperplane.

Algorithm 2 Binary search of critical point using an oracle.

Require: an input $\boldsymbol{X} \in \mathcal{X}$, an oracle $\mathsf{O} : \mathbb{N} \times \mathbb{N} \times \mathcal{X} \times \mathcal{X} \to \mathcal{B}$, the targeted neuron $\eta \in \mathbb{N}$ and a distance precision $\Delta \in \mathbb{R}$
Ensure: $\boldsymbol{X} \in \mathcal{H}_\eta$
  $\boldsymbol{Y} \leftarrow \texttt{random}(\mathcal{X})$ ▷ Generate a random value in $\mathcal{X}$
  $S \leftarrow \mathsf{O}(\eta, \boldsymbol{X}, \boldsymbol{Y})$
  while S is True **do**
    $\boldsymbol{Y} \leftarrow \texttt{random}(\mathcal{X})$
    $S \leftarrow \mathsf{O}(\eta, \boldsymbol{X}, \boldsymbol{Y})$
  end while
  while $||\boldsymbol{X} - \boldsymbol{Y}||_2 > \Delta$ **do**
    $\boldsymbol{M} \leftarrow \frac{\boldsymbol{X}+\boldsymbol{Y}}{2}$
    $S_M \leftarrow \mathsf{O}(\eta, \boldsymbol{X}, \boldsymbol{M})$
    if S_M is True **then** ▷ Implies that $V(\eta; \boldsymbol{M}) = V(\eta; \boldsymbol{X})$
      $\boldsymbol{X} \leftarrow \boldsymbol{M}$
    else ▷ Implies that $V(\eta; \boldsymbol{M}) = V(\eta; \boldsymbol{Y})$
      $\boldsymbol{Y} \leftarrow \boldsymbol{M}$
    end if
  end while
  return $\boldsymbol{X}$

Direct mapping between the critical points and the neuron. Another side advantage of such an oracle is the direct mapping from the critical point to the neuron. Indeed, as mentioned in Subsection 6.1.1, an important limitation of classical cryptanalytic methods is their inability to determine to which neuron a critical point corresponds, resulting in the extraction of unnecessary critical points. This issue is not present in our methodology, as we use the oracle to extract the state of the targeted neuron. We are therefore able to localize which neuron corresponds to the identified critical point. Furthermore, this direct mapping also naturally removes the issues of merging signatures described in [Liu+25].

Extraction of pooling layers. With this oracle, it is therefore possible to search for all the critical points within a layer. Under the assumption that it is then possible to extract the signatures and the scaling factor signs, it is possible to proceed with the next layer in an iterative process. In this case, our methodology is equivalent to subdividing a targeted MLP composed of $L \in \mathbb{N}$ layers into L one-layer-deep neural networks separated at each activation function. The main advantage of this approach is that if the extraction of the signatures is correct in the first k smaller DNNs, then the input of the $k+1$ DNN

is known, and it is possible to extract it. Furthermore, it is possible to leverage modern architectures to target DNNs composed of non-fully connected layers, such as VGG11 [SZ14] or MobileNets [How+17]. In Figure 6.2, we provide a high-level description of the architecture of a VGG11. This model consists of a series of convolutions, followed by a max-pooling layer, and finally a classification block comprising fully connected layers and a softmax function. Due to the presence of the max pooling layers, such an architecture cannot be targeted as a whole by previous cryptanalytical methods [Jag+20; RK19; CJM20].

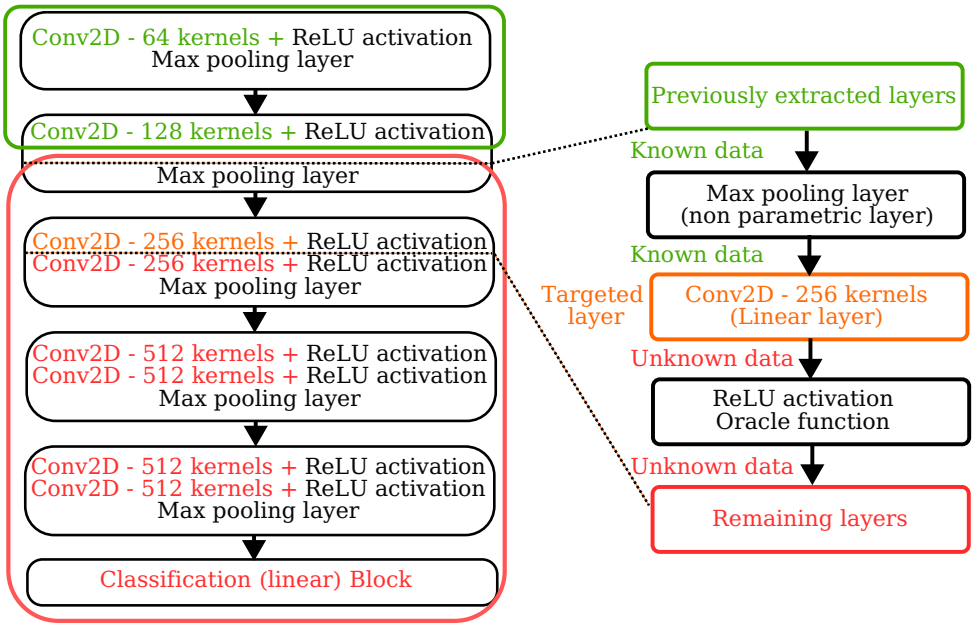


Figure 6.2 – Visualisation of the impact of the subdivision on an architecture incorporating a pooling layer.

By definition, the oracle gives information on the state of the neurons (see Definition 4.7). Consequently, an attacker can obtain internal information at each activation function. Therefore, using the oracle, it is possible to split this architecture after the convolutional layers since these layers are followed by an activation function. Since in such architectures, the pooling layers are directly after an activation function, it is possible to ignore them with the oracle. To illustrate this point, we consider the example of a block k, which corresponds to the subdivision between the second and third layers in a VGG11 model. This block, represented in Figure 6.2, is composed of the following sequence: a

max-pooling layer, followed by a convolutional layer with 256 filters and an activation function. If the previous layers have been successfully extracted, then the input of the block k is known, and since the max-pooling layer is a non-parametric layer, its output is also known. Therefore, in such configurations, the pooling layers can be ignored, as they correspond to known transformations applied to known data. Consequently, complex architectures, including those with pooling layers, such as VGG11, can be targeted. While here we illustrate the example of a VGG11, in SOTA architectures, it is common for pooling layers to be placed directly after an activation function [How+17; SZ14; He+15b], making these architectures potential targets for our methodology.

While we have described the potential benefits of using the oracle to target more complex architectures, we have done so under the assumption that we can extract the signature from the set of critical points and the sign of the scaling factor in hard-label settings. In the following sections, we resolve both of these assumptions. We first present the basic case for signature extraction from the critical points and then introduce how the oracle can also be used to make the extraction process more robust to special cases of neurons. Then, we propose a new method to extract the sign of the scaling factor using the oracle.

6.2.2 Improved Signature Extraction

Since each critical point gives an equation connecting a known input and the unknown weight vector, it is straightforward to construct a system of equations to solve this issue.

Generic method. We infer the equation of the neuron-induced hyperplane (see Definition 4.8) by constructing a linear system based on Equation 4.6 with the critical points obtained from Algorithm 2. Such a system can then be used to extract the weights. Given a set of $n+1$ critical points $\{\boldsymbol{X}^1, \boldsymbol{X}^2, \dots, \boldsymbol{X}^{n+1}\}$ s.t. $\boldsymbol{X}^i \in \mathbb{R}^n$, the following system of equations can be used to extract the n weights and the bias of the targeted neuron η:

$$\begin{cases} \sum_{i=1}^{n} \dfrac{-\boldsymbol{\theta}_{\eta,i}}{\boldsymbol{\theta}_{\eta,0}} \boldsymbol{X}_i^1 - \dfrac{\beta_\eta}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^1 \\ \vdots \\ \sum_{i=1}^{n} \dfrac{-\boldsymbol{\theta}_{\eta,i}}{\boldsymbol{\theta}_{\eta,0}} \boldsymbol{X}_i^{n+1} - \dfrac{\beta_\eta}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^{n+1} \end{cases} \tag{6.1}$$

Once enough critical points have been sampled with Algorithm 2 to form a full rank system, we find the equation of the hyperplane by solving the Least Squares (LSTSQ) problem of this system of equations. We then extract the signature (see Definition 4.9) of the targeted neuron by arbitrarily fixing the scaling factor $\theta_{\eta,0}$ to ± 1. The sign of the scaling factor determines the state of the neuron, and is therefore a critical component to extract. The extraction of the scaling factor's sign will be detailed in Subsection 6.2.3.

It is important to notice that the presence of linear transformations does not impact the proposed method, *e.g.*, batch normalization, between the matrix multiplication and the activation function. Indeed, it is possible to fuse all these linear operations into one and then extract an aggregate of the weights. This fusing is illustrated in Figure 6.3.

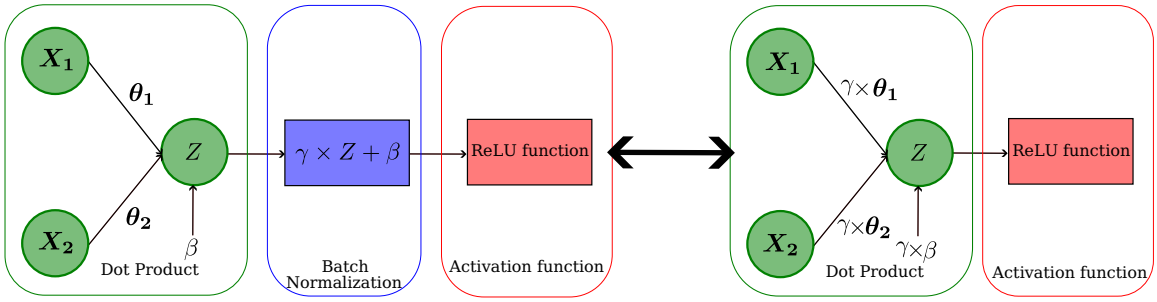


Figure 6.3 – Illustration of the fusion of a linear layer with batch normalization.

This method extracts the signature of a targeted neuron using the critical points obtained from Algorithm 2 under the assumption that a full rank system can be constructed. This implies that the number of critical points, *i.e.*, the number of equations, needs to be at least equal to the number of targeted parameters, *i.e.*, weights and bias, in the neuron, and that each targeted parameter must be expressed through a non-null entry value in at least one critical point.

Unfortunately, there exist neurons for which it is difficult to meet these conditions, and which therefore require a very large number of queries to find a subset of critical points satisfying them. Such neurons can be broadly classified into two types:

- The first type of these neurons is introduced in Subsection 6.1.1 and corresponds to the *always-off* and *always-on* neurons. Such neurons almost never change states, except on a very small subset of the input space $\mathcal{X}$, making it difficult to find critical points via random queries.
- The second type is not reported in the literature and corresponds to neurons for which one of the entries of their critical points is almost always null. For these

neurons, the number of queries needed to form a full-rank matrix can exceed the average number of queries needed for classical neurons by a factor of 10 (see Subsection 6.4.2). We designate such neurons as *input-off*.

The presence of these neurons is correlated with the type of activation functions used. In particular, ReLU functions have been shown to cause the presence of *always-off* neurons [Lu20], and we hypothesize that the nesting of this activation function is the main factor in the occurrence of *input-off* neurons in DNNs. For these types of special neurons, the signature extraction process differs. Once a special case neuron is detected by the abnormal number of queries required during the extraction, a specialized procedure is performed depending on the type of special neuron. Indeed, under the assumption that we are able to extract the sign of a given neuron using our oracle, we can differentiate *always-off* from *always-on* neurons. As the presence of these neurons has been reported in ReLU-based DNNs, we now describe the different processes to extract these neurons based on their types under the assumption that the targeted model uses only ReLU activation functions.

Extraction of *always-off* neurons. The simplest case of a specific neuron is the *always-off* neuron. Since its output value is almost always null and therefore independent from the input, we can directly assign its output value to 0 without having to extract the related weights.

Extraction of *always-on* neurons. For *always-on* neurons, the search for critical points can be extremely long and costly in queries. Furthermore, due to the sequential paradigm of our attack, applying our method on a layer without complete extraction of the previous ones is impossible. It is then crucial to extract all neurons at each layer. Since we know that the neuron is *always-on*, one solution is to consider the neuron's activation function, *i.e*, the ReLU function, as the identity and treat the neuron as a weighted skip connection between this layer and the next one. We illustrate the approximation process for an *always-on* neuron η in the first layer being extracted as part of the signature associated with neuron λ in the following layer in Figure 6.4.

Let $\boldsymbol{X} \in \mathbb{R}^n$ be the input of a layer l composed of m neurons and η an *always-on* neuron in this layer l with $\boldsymbol{\theta} \in \mathbb{R}^n$ (resp. $\beta_\eta \in \mathbb{R}$) its weight vector (resp. bias). Let λ be a neuron in layer $l+1$ and $\boldsymbol{\gamma} \in \mathbb{R}^m$ (resp. $\beta_\lambda \in \mathbb{R}$) its weight vector (resp. bias). If we denote $\boldsymbol{A}_i$ the activation value of the neuron i in layer l, *i.e.*, the input $\boldsymbol{X}$ of layer l, the

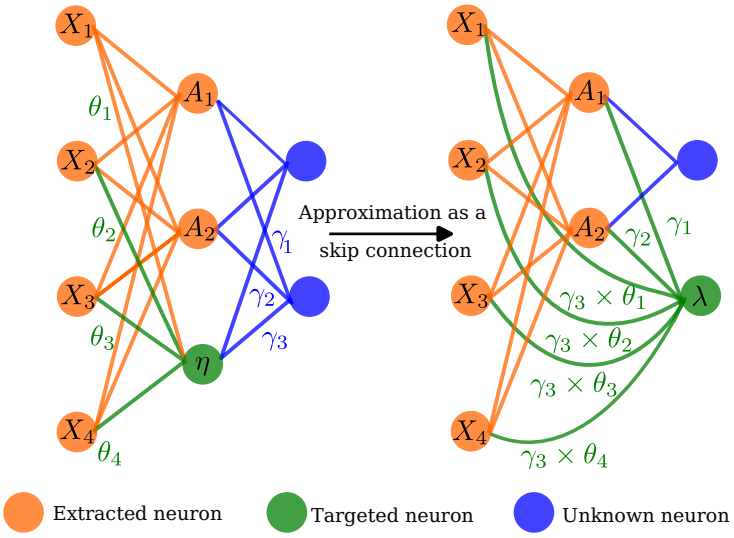


Figure 6.4 – Approximation of the special case neuron η as a skip connection to the neuron λ.

neuron η can be approximated as a skip connection to λ, and Equation 4.6 then becomes

$$\sum_{\substack{i=1,\\ i\neq\lambda}}^{m} \boldsymbol{\gamma}_i \boldsymbol{A}_i + \boldsymbol{\gamma}_\eta \times \sum_{j=1}^{n} \boldsymbol{\theta}_j \boldsymbol{X}_j + \boldsymbol{\gamma}_\lambda \beta_\eta + \beta_\lambda = 0. \tag{6.2}$$

For the example, described in Figure 6.4, Equation 6.2 becomes

$$\sum_{i=1}^{2} \boldsymbol{\gamma}_i \boldsymbol{A}_i + \boldsymbol{\gamma}_3 \times \sum_{j=1}^{4} \boldsymbol{\theta}_j \boldsymbol{X}_j + \boldsymbol{\gamma}_3 \beta_\eta + \beta_\lambda = 0, \boldsymbol{X} \in \mathbb{R}^4. \tag{6.3}$$

As we can see, Equation 6.2 remains linear with respect to the targeted neuron's weights, and a new system of equations can be constructed. Using the same notations as before, we can describe it in the following manner:

$$\begin{cases} -\sum_{i=1}^{m} \dfrac{\boldsymbol{\gamma}_{\lambda,i}}{\boldsymbol{\gamma}_{\lambda,0}\boldsymbol{\theta}_{\eta,0}} \boldsymbol{A}_i^1 - \sum_{j=1}^{n} \dfrac{-\boldsymbol{\theta}_{\eta,j}}{\boldsymbol{\theta}_{\eta,0}} \boldsymbol{X}_j^1 - \dfrac{\beta_{\eta\lambda}}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^1 \\ \vdots \\ -\sum_{i=1}^{m} \dfrac{\boldsymbol{\gamma}_{\lambda,i}}{\boldsymbol{\gamma}_{\lambda,0}\boldsymbol{\theta}_{\eta,0}} \boldsymbol{A}_i^z - \sum_{j=1}^{n} \dfrac{-\boldsymbol{\theta}_{\eta,j}}{\boldsymbol{\theta}_{\eta,0}} \boldsymbol{X}_j^z - \dfrac{\beta_{\eta\lambda}}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^z \end{cases} \tag{6.4}$$

with $\beta_{\eta\lambda}$ an aggregate of the biases of η and λ, and z the number of total equations (s.t.

$z \geq n + m + 1$).

As we can see, the system of equations is very similar to the one described in Equation 6.1. Therefore, we use the same method to solve it. We can also observe that the signature of the neuron η appears in the extracted signature of λ. So, by solving the LSTSQ problem for a neuron in the next layer, it is possible to find the signature of an *always-on* neuron. The only exception is the bias, which cannot be directly extracted. Instead, solving the system of equations returns a combination of the bias from η and λ. However, as the signature extraction of the weights is already performed, only the bias remains unknown. This implies that we only require one additional non-null critical point, corresponding to the targeted *always-on* neuron, to extract the bias.

Extraction of *input-off* neurons. Finally, the case of the *input-off* neurons has a very similar solution to that of the *always-on* neurons. The idea is also to use the signature search for a deeper neuron to extract the signature of the targeted neuron. However, for *input-off* neurons, their state changes with the input, and the activation function can no longer be approximated by the identity function for all inputs. Since we are targeting ReLU activations, we know that the activation function is equivalent to either the null or the identity function. Therefore, we use the oracle under the assumption that we can infer the state of the neuron to detect which inputs lead to the null or identity function, and artificially transform the inputs' equation ourselves. This assumption is coherent with the previous assumption that we made about the possibility of differentiating *always-off* from *always-on* neurons, and will be further described in Subsection 6.2.3. Then we solve the same system of equations as for the *always-on* neurons to extract the neuron's signature. Similarly, to the case of *always-on* neurons, we extract the bias using a critical point directly for the targeted *input-off* neuron.

Using the oracle, it is then possible to extract the signature of a given neuron and even make the process more robust to special cases of neurons. However, both the robustness and the final part of the signature extraction process are under the assumption that the state of the targeted neuron can be extracted.

6.2.3 Estimation of the Sign of the Scaling Factor

The extraction of the sign, using the oracle O, corresponds to a two-step process based on hypothesis testing for all of the neurons included in the subdivided network. While in

DNNs that correspond to a subdivision made by the oracle, several transformations could be applied before the next activation function, and therefore, the next known state. These transformations could all be fused as illustrated in Figure 6.3 as long as they correspond to linear operations. We consider the case of alternating linear and non-linear parametric transformations within a subdivision made by the oracle to be out of the scope of this chapter. Therefore, under this hypothesis, we can consider that all transformations are combined into a single layer, and our extraction of the scaling factor operates on the whole layer. The first part is layer alignment, which drastically reduces the number of hypotheses to one for each layer. And then, we select the correct hypothesis in the next layer.

Layer alignment. The extraction of the sign of all the scaling factors begins once all of the neurons' signatures in the layer have been extracted. In this case, the attacker knows the induced hyperplane for each neuron and can associate an arbitrary state to each side of the hyperplanes, referred to as V_A and V_B, creating 2^n hypotheses, where n is the number of neurons in the layer. To reduce this number, we can align the neurons such that for each neuron in the layer, the arbitrary state V_A corresponds to the same neuron's real state (either active or inactive), while V_B corresponds to the other. The number of hypotheses is reduced to one corresponding to the identification of the sign related to V_A or V_B. This alignment is performed using the oracle O. Indeed, by comparing the states $V(\eta; \boldsymbol{X})$ and $V(\gamma; \boldsymbol{X})$ of two neurons, η and γ, for a given input $\boldsymbol{X}$, it is possible to align the scaling factor associated with the signature of these two neurons.

Hypothesis selection. After the neurons of the targeted layer have been aligned, one assumption is made on the arbitrary state: we suppose that the state V_A (resp. V_B) corresponds to the active (resp. inactive) state. This hypothesis leads to two representations for the whole layer. We test the validity of our hypothesis during the signature search of the first neuron in the next layer. Indeed, once enough critical points are extracted, a linear relation between these points and the output of the previous layer is expected (see Subsection 6.2.2). A wrong assumption on the state V_A and V_B will break this linear relation by having the ReLU function compute $\boldsymbol{A} = \texttt{ReLU}(-\boldsymbol{X})$ instead of $\boldsymbol{A} = \texttt{ReLU}(\boldsymbol{X})$. This produces incorrect activation values, *i.e.*, incorrect inputs for the next layer, and Equation 4.6 no longer holds for the next layer. Therefore, we use the residuals of the solution to the LSTSQ problem (see Subsection 6.2.2) as a criterion for the validity of our sign

hypothesis selection. The correct hypothesis is the one leading to the lowest residual error.

Using this method, we can extract the sign of each neuron without requiring supplementary queries. Furthermore, since our method only requires one hypothesis per layer, we argue that it remains practical even for complex and deep architectures. The final assumption that we need to justify is the ability to differentiate the state of the neurons in the case of special case neurons.

Indentifying the type of special case neuron. In Subsection 6.2.2, we made the hypothesis that we were able to identify the type of special case neurons. By definition, the special case neurons cannot occur on the first layer of the targeted DNN since the attacker crafts the input sent to the DNN. Therefore, once a special case neuron η has been detected in a layer l, it is in a layer such that the previous layer $l-1$ has been completely extracted. Since this layer has been completely extracted, the sign of the neurons is known, and the attacker can use any neuron γ in this layer to test the sign of a given input for both neuron η and γ using the oracle O. As such, depending on the return value of the oracle, the attacker can directly assign the state of a given input to a special case neuron and identify its type.

With this oracle, it is therefore possible to extract the critical points, the signature, and the sign of the scaling factor. Furthermore, this oracle splits the targeted model, enabling the extraction of complex architectures composed of pooling layers. And finally, the last advantage offered by this oracle is that it allows the creation of an alternative and more robust methodology for the signature extraction of special case neurons. In the next section, we introduce the type of oracle that we use and summarize the complete extraction process based on the methodology introduced here and the chosen oracle.

6.2.4 Choice of oracle and complete methodology

Our methodology is based on an oracle O, which returns a boolean value corresponding to *True* if two neurons are associated with the same state for two inputs. Any function that achieves this result can be used as an oracle with our methodology. For example, one could design a methodology based on a fault-injection method, testing the activity of a neuron by observing the absence or presence of variation in the output of a DNN after faulting the pre-activation value in a similar way to what has been introduced in [Hec+23]

for the weights. However, in our case, we use an unsupervised side-channel attack to target the state of neurons using the ReLU as an activation function.

We justify our choice of a side-channel-based oracle by the fact that the activation function of DNNs has already been targeted using a distinguisher exploiting a timing difference in [Bat+19; MBC21], leading to the identification of the used activation function and even the weights' mantissa. While effective, both previously mentioned approaches used supervised learning, and as such, require access to an open device to find the best distinguisher. We consider a more realistic threat model in which an open device is not accessible to the attacker. Therefore, the supervised learning strategy is impossible. To achieve this, we propose another kind of distinguisher based on an unsupervised *k-means* clustering algorithm [DHG01]. The idea is to first generate enough EM traces with random inputs to build a dataset in which both classes, *i.e*, active and inactive neurons, are evenly represented. Then, we use the *k-means* algorithm to identify two clusters corresponding to both states. As mentioned previously (see Section 6.2), it does not matter, at first, to know which cluster corresponds to the active or inactive state, since that information is not needed to find a critical point. As the inputs are randomly generated, these two clusters are then used as a reference to identify the state of a neuron. Finally, given the EM trace of a new neuron, we use the *Euclidean* distance to the two clusters' centroïd as a distinguisher to infer the neuron's state.

Under the assumption that the implementation of the ReLU function is the same for all neurons within all layers, it is then possible to use the distinguisher as an oracle to perform the complete extraction. The complete extraction corresponds to an iterative process over all neurons of all layers, except for the last layer, which is extracted differently due to the absence of an activation function. Our methodology begins by targeting the input layer before moving to the first hidden layer, and so on, as represented in Figure 6.5.

The extraction of each neuron follows the subsequent three stages:

- During the first step of our methodology, we use a side-channel-based oracle to perform a binary search to find the critical points following Algorithm 2. This stage corresponds to the process detailed in Subsection 6.2.1 and is represented in Figure 6.6.

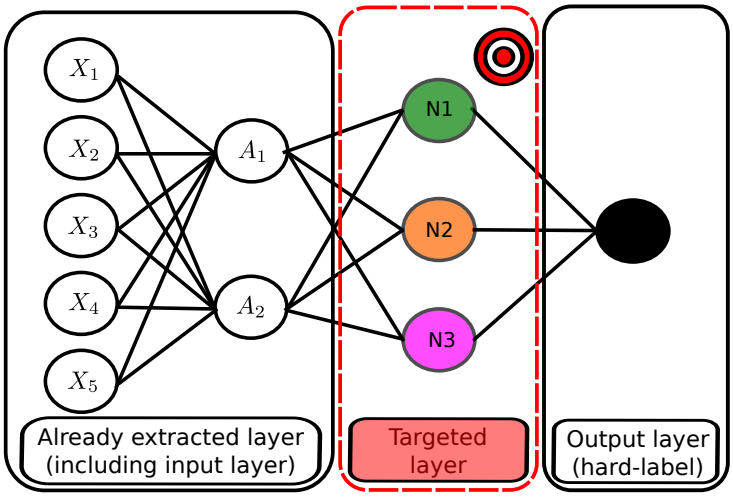


Figure 6.5 – Targeted model extracted in an iterative process.

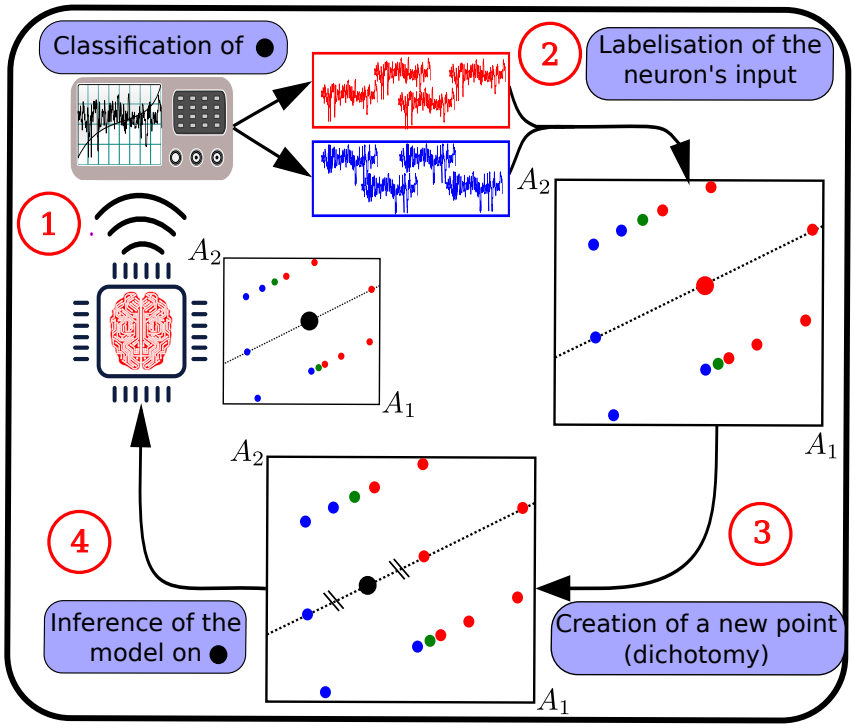


Figure 6.6 – Extraction of the third critical point of neuron N1 of the targeted model using a binary search.

- The second stage consists of retrieving the optimal equation describing the hyperplane induced by the targeted neuron $N1$ from the targeted model represented in

Figure 6.5. The critical points from Stage 1 are used following the method introduced in Subsection 6.2.2. This step is illustrated in Figure 6.7.

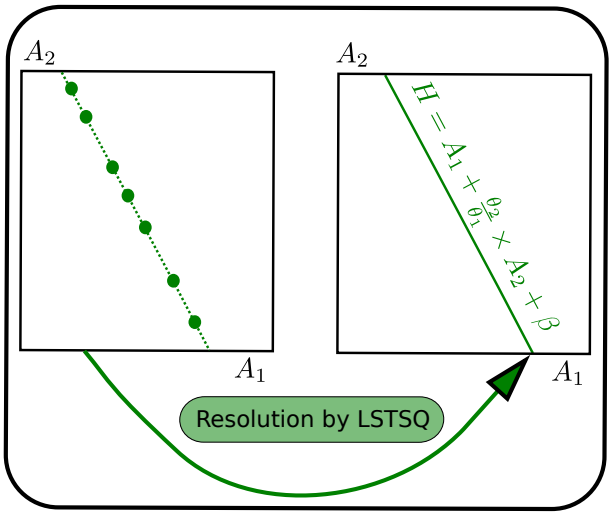


Figure 6.7 – Signature extraction using the critical points from Stage 1.

- In the final stage of our methodology, the sign of each neuron is extracted through hypothesis selection, similarly to [CJM20]. However, as described in Subsection 6.2.3, we use the side-channel-based oracle to align the neurons and reduce the number of hypotheses, as shown in Figure 6.8.

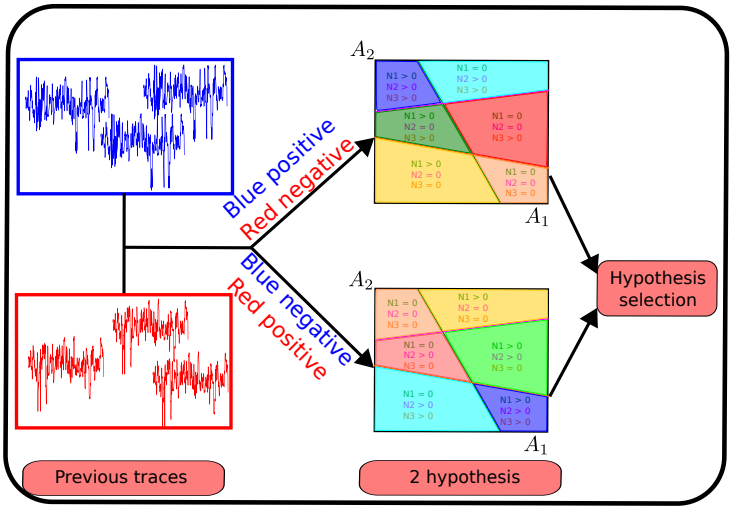


Figure 6.8 – Layer alignment and sign extraction using hypothesis testing.

To validate our methodology, we first test our side-channel-based oracle and its ability to compare the state of two neurons. We then present the results obtained using our methodology on newly targeted architectures and simulations.

6.3 Practical Experiments

6.3.1 Threat Model

As previously mentioned, a wide range of methodologies have been introduced to perform fidelity-based model extraction, and as such, numerous threat models have been considered. In Table 6.1, we summarize the threat model of SOTA methodologies.

Table 6.1 – Overview of the state-of-the-art of fidelity-based model extraction attacks.

Approach	Attack type	Full extraction (weight + bias)	Hard-label setting	Applicable to non-fully connected DNNs	Random queries	DNN's datatype tested	Targeted architecture (Most complex)
[RK19]	Cryptanalytic	✓	✗	✗	✓	64-bit	MLP 10-20-20-1
[CJM20]	Cryptanalytic	✓	✗	✗	✓	64-bit	MLP 40-20-10-10-1
[Che+25a]	Cryptanalytic	✓	✓	✗	✓	64-bit	MLP 1024-2-2-1
[Car+25]	Cryptanalytic	✓	✓	✗	✓	64-bit	MLP 3072-256×3-64-10
[Bat+19]	Side-channel	✗	✓	✓	✓	32-bit	MLP 784-200×4-10
[ZDF23]	Side-channel	✓	✗	✗	✓	64-bit	LeNet5
[Rak+21]	Fault Injection	✓	✓	✓	✗	8-bit	ResNet34 and VGG11
[Hec+23]	Fault Injection	✓	✗	✓	✗	8-bit	CNN 3×(Conv + Pooling + ReLU)-Linear

*Attacks which are **not** in hard-label settings suppose access to the confidence scores.*
*Contributions which are **not** using random queries use a subset of the training or a testing dataset to perform part of their attacks.*

To the best of our knowledge, there is no previous methodology that demonstrates the complete extraction of weights and bias in hard-label settings using random queries without requiring an open device or previous reverse engineering. In the remainder of this chapter, we consider an attacker aiming to extract the weights of a DNN implemented in an embedded system, *e.g.*, an MCU. As countermeasures against side-channel attacks on DL systems are still uncommon, we consider that no countermeasure is implemented on the targeted system. Therefore, the order of the neuron in the activation function is supposed to be constant and known to an attacker. The purpose of this attack is to achieve functional equivalence between the victim's model and the stolen copy (see Definition 4.4). We assume that the attacker has physical access to the device on which the targeted DNN is running. This allows recording of the side-channel traces, *e.g.*, EM traces in our case, during the processing of the input data by the DNN. However, contrary

to [ZDF23; Hec+23], we restrict ourselves to the case where the attacker does not have the possibility to profile the embedded device. As a consequence, it is not possible to train a distinguisher on labeled side-channel traces, and thus, our side-channel-based oracle is non-profiled. Therefore, as stated, our oracle is the *Euclidean* distance to the two clusters' centroïd generated by the unsupervised clustering algorithm *k-means*.

Similar to [Jag+20; RK19; CJM20], we consider only Deep-ReLU networks as targets in this threat model, and we assume that the exact architecture of the targeted model is known. However, we do not restrict ourselves to fully connected layers. Finally, we suppose unrestricted query access to the model embedded on the targeted device. However, in this threat model, we examine the case where the output of the targeted DNN is restricted to the hard label, rather than a vector composed of the soft labels or the logits. In conclusion, we propose a new non-profiled gray-box attack to perform fidelity-based model extraction of complex DNNs in hard-label settings.

6.3.2 Experimental Setup

Dataset and Architecture. We target several embedded DNNs, including a shortened MobileNetv1 [How+17]. We provide an exact description of this model in Table 6.2. This DNN shares the same basic blocks as the original MobileNetv1: depth-wise separable

Table 6.2 – Description of the architecture of the shortened MobileNetv1.

Layers	**Output size**	**Parameters**
Conv2D 3×3, padding, stride=2	$1 \times 12 \times 16 \times 16$	324
BatchNorm2D	$1 \times 12 \times 16 \times 16$	24
ReLU	$1 \times 12 \times 16 \times 16$	0
Depth-wise separable convolution	$1 \times 8 \times 16 \times 16$	244
Depth-wise separable convolution	$1 \times 16 \times 8 \times 8$	248
Depth-wise separable convolution	$1 \times 16 \times 8 \times 8$	464
Depth-wise separable convolution	$1 \times 32 \times 4 \times 4$	752
Depth-wise separable convolution	$1 \times 64 \times 4 \times 4$	2528
Adaptive Average Pooling	$1 \times 64 \times 1 \times 1$	0
Dense	1×10	650

convolutions. The architecture of a depth-wise separable convolution is described in Table 6.3. To the best of our knowledge, these blocks have never been successfully extracted using cryptanalytic extraction. We also tested our methodology against a large MLP with the architecture described in Table 6.4.

Table 6.3 – Description of a depth-wise separable convolution taking N_c channel as input.

Layers	**Kernel size**	**Padding**	**Grouped convolution**	**Bias**
Conv2D BatchNorm2D ReLu	3×3	1	✓	✗
Conv2D BatchNorm2D ReLu	$N_c \times 1 \times 1$	1	✗	✗

Table 6.4 – Description of the architecture of the large MLP.

Layers	**Output size**	**Parameters**
Flatten	1×3072	0
Dense + ReLU	1×512	$1,573,376$
Dense + ReLU	1×256	$131,328$
Dense + ReLU	1×64	$16,448$
Dense + ReLU	1×10	650

To ease the description of the targeted architecture, in the case of MLPs, we will use the following nomenclature to describe the architecture: Size Input - (Size Hidden Layer) - Size Output Layer. For example, the large MLP previously described is noted: 3072-512-256-64-10. Finally, we target the most complex architectures included in [Jag+20; RK19; CJM20; Che+25a; Car+25]. We summarize the targeted architectures in Table 6.5.

Table 6.5 – Summary of the targeted architectures.

Type of architecture	**Nb. of parameters**	**Nb. of targets**	**Targeted by**
Small MLP	$< 200,000$	7	[CJM20; Jag+20; RK19; Che+25a], This work
MLP	$935,370$	1	[Car+25], This work
Large MLP	$1,721,802$	1	This work
MobileNetv1-short	$5,234$	1	This work

All classifiers are trained on the CIFAR-10 dataset [1] except for the architecture from [Che+25a]. This one, as well as all the DNNs performing regression, is trained on a random dataset, as in the SOTA.

1. `https://www.cs.toronto.edu/~kriz/cifar.html`

Hardware configuration. The models are embedded on an STM32-F767ZI board, based on an ARM Cortex-M7 MCU, via the X-Cube-AI framework without optimization from the compiler, and use 32-bit floating-point data for all processed values, *e.g.*, input, weights, activation values. This board incorporates 2 MBytes of Flash memory and 512 KBytes of SRAM. X-Cube-AI is a framework developed by ST Microelectronics to facilitate the deployment of DNNs on STM's embedded devices. However, it is important to note that this framework is not intended to provide secure deployment for embedded DNNs.

Practical setup. A probe from Langer (EMV-Technik RF-U 2.5) is connected to an amplifier (ZLF-2000G+) to capture the EM traces. To acquire the EM signal, a Lecroy oscilloscope (2.5 GHz WaveRunner 625Zi) is used in our setup. To ease the leakage exploitation, a trigger surrounding the processing of the ReLU function has been added. Since the patterns surrounding the ReLU function are associated with the matrix product, they are distinguishable from the ones associated with the processing of the ReLU function, and the configuration of the trigger is not considered difficult.

6.3.3 Side-Channel Analysis of the ReLU Implementation

Our methodology is completely based on the oracle and its ability to distinguish the state of two inputs for a given neuron. Therefore, we first validate that we can create such a function using our distinguisher function introduced in Subsection 6.2.4. In this section, we present our results using the setup introduced in the previous section.

Description of the vulnerability. The implementation of the ReLU function in X-Cube-AI is constant-time, as recommended in [MBC21]. Algorithm 3 describes the main idea behind this implementation [2] for 32-bit values. Given a signed pre-activation value

Algorithm 3 Pseudo-code of the constant-time ReLU function.

Require: Pre-activation value $z \in \mathbb{R}$
Ensure: $a = \texttt{max}(0, z) \in \mathbb{R}^+$
$a \leftarrow (\sim (z \gg 31)) \ \& \ z$

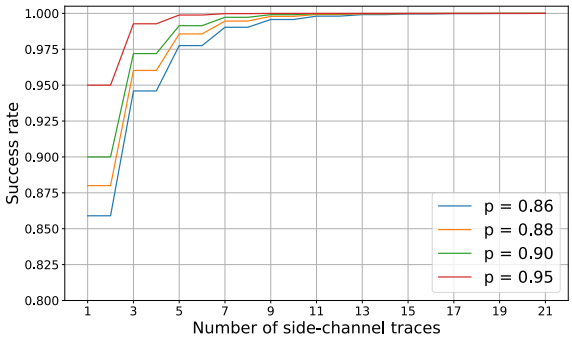
2. For the purpose of this work, we decided to explain the weakness we exploit as an illustrated example, but understanding the vulnerability is not a prerequisite to performing our black-box extraction attack.

$z \in \mathbb{R}$ encoded on 32 bits, the sign can be extracted ($z \gg 31$), and cast on 32 bits to serve as a mask, *i.e.*, $(\sim (z \gg 31)) \in \{$`0x00000000, 0xFFFFFFFF`$\}$. Hence, depending on the sign of z, application of the mask will either cause no modification or set the output to zero. Although this implementation is constant-time, the usage of the mask leads to a physical leakage that can be exploited by an attacker. In particular, depending on the state of the neuron, the mask is only defined by `0x00000000` or `0xFFFFFFFF`. The difference between the Hamming weights of these two values allows the creation of two groups on the side-channel traces, using clustering algorithms, corresponding to the two active and inactive states.

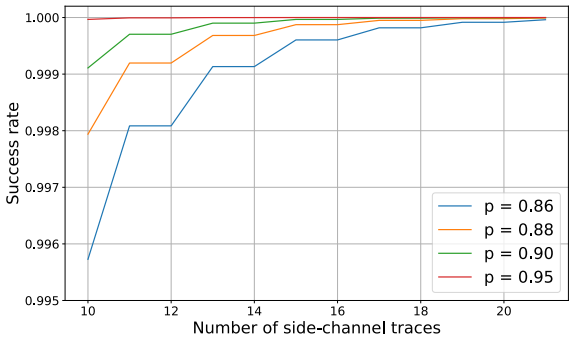
Critical point search. To validate our method, we acquire a set of 10^4 EM traces using the setup defined in Subsection 6.3.2. Then, we conduct the strategy defined in Subsection 6.2.4 and measure the success rate of retrieving the neuron's state for 12 neurons. We use the *k-means* algorithm to generate the two neurons' state clusters without any prior knowledge of the neurons' states themselves. As stated in Subsection 6.2.4, we only need a correct separation between the two clusters for our attack. We measure the success rate in retrieving the correct classification, *i.e.*, that two neurons with the same state are classified in the same cluster. Therefore, to compute this metric and verify the suitability of the *k-means* algorithm, we suppose access to the true state of the neurons.

We achieve an average success rate of 85.9%, using only one trace. While this result validates our methodology, we expect that the high number of queries needed to perform the attack (we share the same query complexity as [Jag+20], see Subsection 6.2.1) imposes a higher success rate (close to 100 %) to extract complex DNNs. A basic solution to increase the success rate would be to collect more traces of the processing of the same input and use a majority vote to infer the state.

In Figure 6.9a, we plot the success rate obtained with a majority vote with respect to the number of acquired traces for different initial success rates, *i.e.*, the success rate in one trace. With an initial success rate of 85.9%, we now achieve a success rate of over 99% with 7 traces and over 99.9% with 13 traces. The success rate and the number of side-channel traces required to achieve this level are highly dependent on several experimental factors, such as the overall noise generated by the physical system and the type of probe, for example. Given the expected high number of queries required to fully extract a DNN, it is crucial to maintain a high success rate while minimizing the number of needed traces. To consider the worst-case scenario for the victim, *i.e.*, best-case scenario for the attacker,



(a) Success rate with different initial probabilities.



(b) Zoom of Figure 6.9a after ten traces in the same settings.

Figure 6.9 – Evolution of the success rate of the extraction of the state of a neuron as a function of the number of side-channel traces.

we suppose an ideal setup allowing the attacker to perform side-channel attacks with a 100% success rate in only one EM trace. In the following sections, all of the results are given under this hypothesis. This allows us to establish an empirical boundary for the performance of our methodology, which will facilitate future comparisons with other methodologies. Finally, this assumption is not restrictive, since if one attack requires N traces to achieve a 100% success rate, the global complexity of the attack is given by the worst-case scenario complexity multiplied by N.

Sign extraction. The last property that we need to validate for the oracle is the capacity to align the state of each layer to perform sign extraction. For this step, we exploit the method introduced in Subsection 6.2.3 to reduce the number of clusters to two for the whole layer. Based on this method, we achieve a success rate of 99.7% in retrieving the sign of the scaling factor related to each neuron within one trace.

Our results are not dependent on the algorithm and could be improved using different unsupervised methods. However, the overall principle will remain the same. Furthermore, since the vulnerability originates from the software implementation, any platform using this implementation is a potential target. As our research indicates, this is the first successful extraction of the neuron's sign in a hard-label setting that does not require extra queries.

6.3.4 Complete Extraction of Classifiers

To evaluate the performance of our methodology, we use the metrics introduced in Subsection 4.3.1, fidelity, *i.e.*, label agreement, AuA (see Definition 4.6), and the maximum of the absolute difference between the correct weights θ and their extracted counterparts $\hat{\theta}$. We compute this metric both on the whole network of depth L ($\texttt{max}|\Delta_\theta|^L$) and on the first $L-1$ layers ($\texttt{max}|\Delta_\theta|^{L-1}$). Finally, to evaluate the cost of our attack, we use the number of queries made to the targeted model.

Summary of the complete results. To the best of our knowledge, cryptanalytical extraction of DNNs in hard-label settings has only been performed in [Che+25a; Car+25]. Therefore, we first test our methodology on the largest architectures from these previous works, a 1024-2-2-1 MLP [Che+25a] and a 3072-256-256-256-64-10 MLP [Car+25], respectively referred to as MLP-a and MLP-b. However, we would like to highlight that, whereas the methodology proposed in [Car+25] allows for cryptanalytic extraction of the hidden layers, they do not perform the extraction of the last layer. As a result, it is not possible to compute metrics for the full extracted network, *i.e.*, number of queries, fidelity, and $\texttt{max}|\Delta_\theta|^L$. We also extract a 3072-512-256-61-10 MLP (referred to as MLP-c) composed of 1.7 million parameters, nearly doubling the previous maximum number of extracted parameters in SOTA studies. We summarize the results in Table 6.6.

Table 6.6 – Comparison in hard-label settings.

Architecture	Parameters	Approach	Queries	$\texttt{max}\|\Delta_\theta\|^{L-1}$	$\texttt{max}\|\Delta_\theta\|^{L}$	Fidelity
MLP-a 1024-2-2-1 (64-bit data)	**2,059**	[Che+25a]	$2^{22.49}$	-	$\mathbf{2^{-20.38}}$	**100%**
		This work	$\mathbf{2^{17.12}}$	$\mathbf{2^{-42.5}}$	$2^{3.45}$	99.5%
MLP-b 3072-256-256-256 -64-10 (64-bit data)	**935,370**	[Car+25]	-	$2^{-18.0}$	-	-
		This work	$\mathbf{2^{26.2}}$	$\mathbf{2^{-26.6}}$	$2^{2.1}$	97.2%
MLP-c 3072-512-256-64-10 (32-bit data)	**1,721,802**	**This work**	$2^{26.0}$	$2^{2.3}$	$2^{3.2}$	93.2%
Truncated MobileNetv1 (32-bit data)	**5,234**	**This work**	$2^{18.8}$	$2^{-4.3}$	$2^{3.7}$	88.4%

Performance comparison. We successfully extract the previously mentioned architectures with 99.5% (resp. 97.2%) for MLP-a (resp. MLP-b). Compared to the methodology proposed in [Che+25a], we improve the cost of the attack by dividing the total number of necessary queries by a factor of ≈ 41, while offering a similar level of fidelity with 100% [Che+25a] and 99.5% for our approach. This drop in the fidelity is due to the fact

that their methodology is specifically designed for binary classifiers. Indeed, for binary classifiers, it is possible to form a system of equations using decision boundary points, *i.e.*, points on the boundary where the output label changes, and the activation value of the previous layer. While this allows for precise extraction of the last layer, it is not directly applicable to all multiclass classifiers. Therefore, we use our method, which is more generic.

Since Carlini *et al.* in [Car+25] did not perform the complete extraction of MLP-b, we can only evaluate our methodology on the maximum error on the hidden layers' weights. Using this metric, we observe that our methodology offers a more accurate estimation of the weights by a factor of 388. Furthermore, in their work [Car+25], they suppose a perfect extraction of the previous layer when targeting a new one, which removes the propagation of residual errors. For comparison, we also simulated our methodology on this model under the assumption that the extraction of the previous layer is perfect. We compute the maximum absolute difference between the weights at each layer with and without this assumption and include the results in Table 6.7. Under this hypothesis, we

Table 6.7 – Maximum error on the weights with and without the residual error on the MLP architecture from [Car+25]

Assumption	**Layer 1**	**Layer 2**	**Layer 3**	**Layer 4**	**Layer 5**
With residual error	$2^{-38.1}$	$2^{-33.8}$	$2^{-29.7}$	$2^{-26.6}$	$2^{2.1}$
Without residual error	$2^{-38.1}$	$2^{-41.9}$	$2^{-42.8}$	$2^{-41.3}$	$2^{2.0}$

can see that the performance is not impacted by the depth of the DNN, and the extraction proposed by our methodology offers similar performance at each layer. In consequence, in this configuration, *i.e.*, without error propagation, we further improve the results obtained in [Car+25] by dividing the maximum error on the weights of the hidden layers by a factor of $\approx 2^{20}$.

Finally, even when considering error propagation, we demonstrate the complete extraction of the two new architectures with high fidelity. We achieve a fidelity of 88.4% (resp. 93.2%) on the complete testset of CIFAR-10 ($10,000$ images) for the MobileNetv1 model (resp. MLP-c).

Comparison with *learning-based* extraction. We compare our results with *learning-based* extractions using a baseline model trained on a balanced dataset composed of the queries used during the critical points' search, with the responses of the targeted model as

labels. We use the same hyperparameters, *e.g.*, architecture, learning rate, as for the targeted model. While the baseline methodology appears to offer decent performance during training (around 56% of accuracy on the training dataset), it does not generalize well, and the performance drops significantly on the test dataset, achieving only a fidelity of 21.1% for the MobileNetv1. The use of random queries, which are not from the same distribution as the true training set from CIFAR-10, and the hard-label setting drastically reduce the performance of the baseline model. This illustrates that *learning-based* extraction methods are limited by data distribution, which makes cryptanalytic attacks more appropriate for offering better performance in the threat model considered in this chapter.

Generation of adversarial examples. To finalize our performance evaluation, we use AuA to quantify how much our extraction attack can improve other attacks, and more specifically, adversarial attacks in the settings described in [Rak+21; Hec+23]. We craft adversarial examples using the stolen model and measure their impact on the targeted model. We obtain transfer rates of 98.63%, 96.67%, and 95.78% for MLP-a, MLP-c, and MobileNetv1, respectively, thus achieving close to white-box performance. Our methodology outperforms a powerful substitute model trained with access to the true data [Rak+21] and proves its ability to mount more powerful attacks on the targeted model in hard-label settings.

6.3.5 Simulation of our Methodology in Soft-Label Setting

To ensure a better comparison between our methodology and other cryptanalysis-based methodologies, we include results from SOTA on DNNs trained for regression tasks, assuming access to the continuous output value. In this context, we take advantage of this access to improve the extraction of the last layer, which differs from the previous layers as mentioned in Subsection 6.2.4. On the other hand, previous methodologies use this value to extract the hidden layer directly and, consequently, can perform gradient-based optimisations to improve the efficiency of the extraction process. However, none of these works can target DNNs in hard-label settings, unlike our methodology.

For regression tasks, the metrics AuA and fidelity, *i.e.*, label agreement, are no longer meaningful. Consequently, to evaluate the fidelity of the extracted models, SOTA methodologies only consider the maximum error on the extracted weights [CJM20; Jag+20; RK19]. This metric has the advantage of providing an estimate of fidelity independent of any assumption about the input distribution, unlike a layer-wise mean squared error.

Finally, we continue to use the number of queries to quantify the cost of the attack.

As our methodology is not based on the gradient of the targeted DNN, it is not impacted by the precision in the output format. Previous methodologies target DNNs using 64-bit data. In this section, we first present the results obtained from simulations using 64 bits and then provide the results obtained with 32-bit data. We use the same notation to describe the architecture as the one introduced in Subsection 6.3.2, *i.e.*, a 784-128-1 DNN corresponds to a model with an input size of 784, one hidden layer composed of 128 neurons, and 1 output.

Results on DNNs using 64-bit data. Table 6.8 provides a comparison of DNNs trained for regression in soft-label settings with 64-bit data. In terms of fidelity of extraction, our methodology outperforms all previous methods by dividing the maximum error on the weights by a factor of at most $2^{16.1}$ and at least $2^{4.5}$, despite operating in hard-label settings for the extraction of the hidden layers. The side-channel attack offers higher precision in localizing critical points compared to gradient-based methods, which, in turn, provides a more accurate estimation of the weights. Regarding the cost of the attack, our methodology requires fewer queries for four out of the six previously targeted architectures. For the two architectures where our attack is more costly, we hypothesize that this is due to their narrow topology, *i.e.*, 1-layer deep, which seems to improve gradient-based optimizations implemented in SOTA methodologies. For the rest of the architectures, the direct mapping from a critical point to a neuron provided by the side-channel attack improves the efficiency of our methodology compared to the SOTA.

Results on DNNs using 32-bit data. Table 6.9 provides a comparison of our methodology against targeted architectures by SOTA methodologies using 32-bit data. Despite the absence of current comparisons, we provide the following results for future comparison on DNNs using 32−bit data. We successfully extract all architectures in these settings as well.

While our methodology offers strong results, we observe in Table 6.8 and Table 6.9 that the depth and the number of neurons per layer impact the performance of the attack. In the next section, we further analyse this limitation and evaluate the impact of special case neurons as well as the impact of extracting the last layer.

Table 6.8 – Comparison of DNNs trained for regression in soft-label settings with 64-bit data.

Architecture	Parameters	Approach	Queries	$\max\|\mathbf{\Delta}_\theta\|^L$
10-10-10-1	210	[CJM20]	$2^{16.0}$	$2^{-36.0}$
		[RK19]	$2^{22.0}$	$2^{-12.0}$
		This work	$\mathbf{2^{15.6}}$	$\mathbf{2^{-46.2}}$
10-20-20-1	620	[CJM20]	$2^{17.1}$	$2^{-37.0}$
		This work	$\mathbf{2^{15.6}}$	$\mathbf{2^{-46.5}}$
40-20-10-10-1	1,110	[CJM20]	$2^{17.8}$	$2^{-27.1}$
		This work	$\mathbf{2^{16.8}}$	$\mathbf{2^{-42.0}}$
80-40-20-1	4,020	[CJM20]	$2^{18.5}$	$2^{-39.7}$
		This work	$\mathbf{2^{18.3}}$	$\mathbf{2^{-44.2}}$
784-32-1	25,120	[CJM20]	$2^{19.2}$	$2^{-30.2}$
		[Jag+20]	$\mathbf{2^{18.2}}$	$2^{-1.7}$
		This work	$2^{20.6}$	$\mathbf{2^{-43.5}}$
784-128-1	100,480	[CJM20]	$\mathbf{2^{21.5}}$	$2^{-24.7}$
		This work	$2^{22.6}$	$\mathbf{2^{-40.8}}$

Table 6.9 – Results of our methodology against targeted architectures by SOTA methodologies using 32-bit data.

Architecture	Parameters	Queries	$\max\|\Delta_\theta\|^L$
10-10-10-1	210	$2^{13.0}$	$2^{-18.2}$
10-20-20-1	620	$2^{14.5}$	$2^{-17.8}$
40-20-10-10-1	1,110	$2^{16.4}$	$2^{-12.1}$
80-40-20-1	4,020	$2^{19.1}$	$2^{-14.8}$
784-32-1	25,120	$2^{19.8}$	$2^{-17.7}$
784-128-1	100,480	$2^{21.7}$	$2^{-17.4}$

6.4 Discussion and Limitations

6.4.1 Propagation of Errors

As mentioned in Subsection 6.2.1, using binary search, it is only possible to place a critical point within a certain distance of the hyperplane. This creates a small error in the estimation of the weights, as illustrated in Table 6.10, which provides the layer-wise statistics of the shortened MobileNetv1 extraction. The propagation of this small error on the weights in the current layer will be accumulated in the activation values, *i.e.*, the inputs for the system of equations of the next layer, which reduces the performance of the extraction of the next layers. This pattern is clearly visible in Table 6.10. For the first layer L0, the maximum error on the estimated weights is $2^{-18.9}$. While for the last hidden layer,

Table 6.10 – Layer-wise statistics of the extraction of the shortened MobileNetv1.

Datatype	Metrics	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
32 bits	$\texttt{max}\lvert\theta - \hat{\theta}\rvert^L$	$2^{-18.9}$	$2^{-17.6}$	$2^{-7.9}$	$2^{-18.2}$	$2^{-7.6}$	$2^{-13.9}$	$2^{-9.9}$	$2^{-11.4}$	$2^{-8.8}$	$2^{-6.7}$	$2^{-4.3}$	$2^{3.7}$
64 bits	$\texttt{max}\lvert\theta - \hat{\theta}\rvert^L$	$2^{-46.6}$	$2^{-43.8}$	$2^{-37.4}$	$2^{-34.2}$	$2^{-29.0}$	$2^{-27.1}$	$2^{-26.0}$	$2^{-26.8}$	$2^{-23.1}$	$2^{-22.7}$	$2^{-15.3}$	$2^{3.8}$

this error increased to $2^{-4.3}$. For comparison, we also include the result on the propagation of this error using 64-bit data. We observe the same pattern, the only difference being that since we are able to localize the critical points with higher precision, the error after the first layer is much lower, $2^{-46.6}$, compared to 32-bit data, which leads to a lower final error of $2^{-15.3}$. This error propagation is not intrinsic to our method, as all SOTA cryptanalytic extraction methodologies are impacted by it. It is then crucial to minimize the error at each layer to have the smallest impact on the next layer, which underlines the benefits of using an oracle-based methodology. Alternatively, several methods could be implemented to reduce this propagation, such as re-computing the weights to achieve higher precision. However, we leave this improvement for future work.

6.4.2 Impact of the Special-Case Neurons

Another point of interest is the impact of the special-case neurons. Through the oracle, it is possible to improve the robustness of the extraction process, but its extraction remains costly in terms of queries. To visualize the impact of the special cases of neurons, we present their share in the total number of requests and neurons for the 3072-256-256-256-64-10 MLP in Figure 6.10. While previous works mention the existence of *always-off* and *always-on* neurons, their presence is not considered during the extraction process, and existing methodologies do not propose any solutions regarding these specific neurons. This is especially true for *input-off* neurons, which are not even acknowledged in the SOTA. However, we observe the occurrence of such neurons in our experiment, even for simple architectures. In particular, we compute the proportion of queries dedicated to special case neurons per layer as well as the proportion of these neurons per layer, as shown in Figure 6.10 for the 3072-512-256-64-10 MLP. While most of these special neurons correspond to *input-off*, and as such, previous methodologies should be able to extract them, this greatly increases the cost of the attack compared to methods using an oracle to optimize their extraction.

These results highlight the impact of the special case neurons on the number of requests

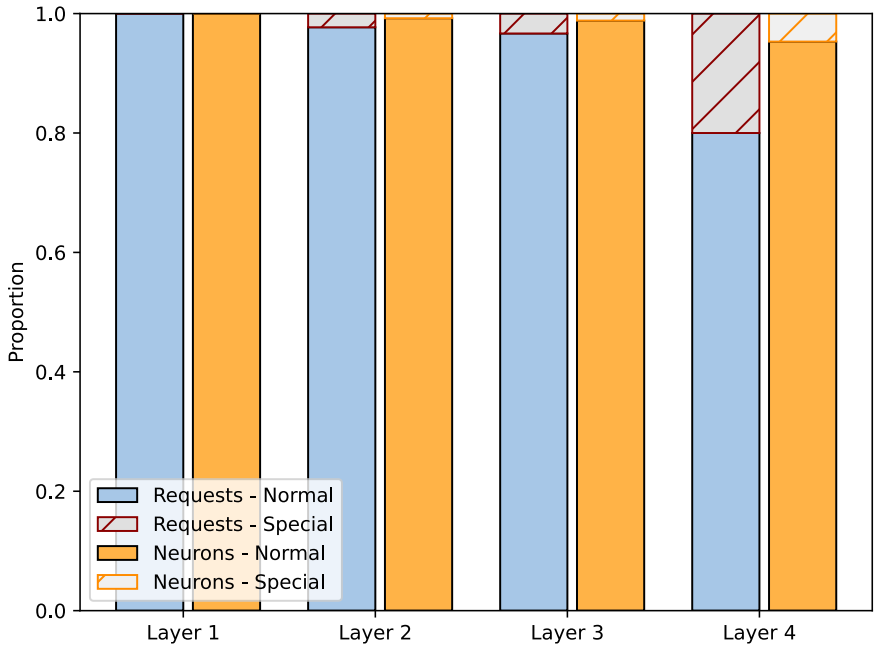


Figure 6.10 – Comparison of the proportion of requests dedicated to normal versus special-case neurons.

to extract a DNN, even for a simple architecture like an MLP composed of five layers.

6.4.3 Extraction of the Last Layer

As previously mentioned, the extraction is different for the last layer. This is why we choose to include both the maximum error on the first $L-1$ layers and on all L layers, *i.e.*, the full network. Indeed, since there is no non-linear activation function for this layer, no critical points can be found. In [Che+25a], the authors use the fact that they target binary classifiers to precisely extract the last layer. On the other hand, to improve the generality of our methodology, we decide to use supervised learning on a dataset composed of the random inputs generated for the previous layers and their corresponding hard-labels, to train the last layer to behave similarly to the last layer of the targeted model. While this methodology allows the extraction of the last layer in hard-label settings, it accounts for most of the error in our process.

To validate this result, we use the first eleven layers that we extract and combine them with the true final layer of the targeted model. This hybrid model achieves a near-perfect fidelity of 99.56%, which confirms that the main loss in the fidelity is coming from the

extraction of the last layer. As this limitation impacts all methods based on the extraction of the critical points in hard-label settings on multi-classifier, we leave the improvement of the extraction of the last layer for future work. Two alternative methods could be used to mitigate this issue. The first one could be to use the methodology presented in Chapter 5 and perform a side-channel attack to retrieve the confidence scores and then extract the last layer. However, this would require a profiling device (see Subsection 5.3.1), and therefore, change the threat model of the complete attack. The other alternative would be to use the methodology introduced in [CS25], *i.e.*, fix several parameters, and extract the last layer under these constraints. Indeed, this approach could directly be used as a supplementary attack to our work to extract the final layer and improve the fidelity of the extraction. However, while this methodology achieves a high label agreement, the last layer is not completely extracted. And since the impact on further attacks, such as evasion attacks, has not been investigated by the authors of [CS25], it should be considered in future work.

6.5 Conclusion

In this chapter, we describe how side-channel attacks can be used as an oracle to improve fidelity-based model extraction attacks. We design a new methodology, inspired by cryptanalysis, but introduce a new paradigm: divide and conquer. With this new approach, we demonstrate the extraction of new complex DNNs, including pooling layers, in hard-label settings while improving the performance of the extraction process. Finally, we validate our approach by simulating the oracle and performing the extraction of the most complex architecture targeted in soft-label settings, and show that despite a stricter threat model, our approach remains competitive and even outperforms previous methodologies in terms of accuracy on the extracted weights.

Several potential future works can be considered to improve our methodology. The first type concerns the oracle. While we have used unsupervised clustering on the EM traces corresponding to the processing of the ReLU function on an MCU, our oracle methodology is neither restricted to this algorithm, *i.e.*, *k-means*, this activation function, nor this type of physical device. Therefore, our methodology could be extended to DNNs using any kind of activation function provided that it is possible to construct an oracle that has the properties described in Subsection 6.2.1. Another challenge will be to search

for an oracle adapted to parallelized architectures such as GPUs, TPUs, or FPGAs. As previously mentioned, in these systems, the processing is parallelized, and as such, the side-channel leakages may be difficult to exploit. One possibility would be to completely change the type of side-channel by using time-to-digital converters, as in [Lom+25], or to consider alternative types of physical attacks and fault injection methods. The second type concerns the adaptation of the methodology to other types of DNN architecture. While in our experiment, we target a MobileNetv1 and several MLPs, a large variety of architectures have not been targeted, and could be considered under the assumption that the correct oracle has been identified.

Finally, the most important adaptation, from our point of view, is the capacity of our methodology to target quantized DNNs. In Subsection 6.4.1, we describe how the error propagates from one layer to another and how the datatype used impacts the original error. However, to the best of our knowledge, no previous work has studied the impact of quantization on cryptanalytical extraction. We expect that using lower precision will further degrade the original error and, as such, limit the extraction of deep architecture. Therefore, cryptanalytical extraction as previously described in [CJM20; RK19; Car+25], and even the methodology proposed in this chapter, may not be applicable to quantized models. For these reasons, we propose introducing new attacks in the following chapters that are adapted to such models. While this chapter and the previous one demonstrate the complementarity of software and hardware attacks by adapting typical software attacks with side-channel analysis, the rest of the thesis will introduce side-channel attacks that are decorrelated from software attacks. We begin by examining the impact of quantization on cryptanalytical extraction in Chapter 7 and propose using side-channel attacks to mitigate this issue. Finally, in Chapter 8, we introduce a template attack to extract the weights of quantized DNNs.

Part III

Introducing New Specific Side-Channel Attack Against the Confidentiality of Deep Neural Networks

Chapter 7

Fidelity-Based Model Extraction of Quantized Deep Neural Networks

We begin this chapter by evaluating the impact of quantization on cryptanalytical extraction of DNNs. As mentioned in Chapter 6, the type of data used directly impacts the error between the extracted and the true weights. However, we report in this chapter the existence of a threshold below which extraction methods based on cryptanalysis fail. We experimentally link this threshold to the quantization of the input and show that it leads to an incorrect estimation of the critical points, resulting in a biased solution. Furthermore, the iterative nature of cryptanalytical methods causes errors in an early layer to accumulate and increase at each successive layer. Therefore, an adaptation is required to perform a complete attack on quantized models. We propose an alternative method allowing the extraction of DNNs independently of their quantization. While the methodology introduced in Chapter 6 corresponds to a new approach to improve cryptanalytical extraction, here we propose a different method, based solely on a side-channel attack, which is closer to the works introduced in [Rak+21; Hec+23] but originating from our observations on cryptanalytical extraction.

Contents

7.1 Impact of Quantization on Cryptanalytical Model Extraction 136

7.1.1 Simulation of the Extraction of Quantized Neurons 136

7.1.2 Impact of the Input Quantization 138

7.2 Active learning with a fidelity objective 142

7.2.1 Moving from equalities to inequalities 142

7.2.2 Global Algorithm 144

7.2.3 Critical vs non-critical inequalities 148

7.2.4 The Different Levels of Fidelity-Based Model Extraction Attacks 149
7.3 Practical Experiments 151
7.3.1 Experimental Setup and Threat Model 151
7.3.2 Analysis of the Side-Channel Vulnerability 152
7.3.3 Results with a Perfect Oracle Function 155
7.3.4 Considering Errors in the Oracle Function 157
7.4 Discussion and Perspectives 160
7.4.1 Comparison with Different Threat Models 160
7.4.2 Inconclusive Alternatives to Inequalities 161
7.5 Conclusion 162

7.1 Impact of Quantization on Cryptanalytical Model Extraction

7.1.1 Simulation of the Extraction of Quantized Neurons

To visualize the impact of quantization on cryptanalytical extraction, we first use an empirical approach and simulate the attack on a set of quantized perceptrons. There are several different quantization targets, *i.e.*, variables that can be quantized, in a perceptron, *e.g.*, weights, bias, input, and activation. We represent the different targets of quantization in Figure 7.1. We first decide to use quantized weights, bias, and inputs to consider full quantization of the perceptron except for its output. At each quantization level, we average the results across a hundred perceptrons, resulting in a total set of $3,000$ perceptrons, each taking a vector of 128 values as input. For the simulation, we use the Brevitas framework [FPF25] to quantize the perceptrons using QAT (see Subsection 2.2.3).

In a similar manner to what is done in [CJM20; Can+24], we assess the success of the extraction by measuring the maximum absolute difference between the extracted weights and the real ones, *i.e.*, $\texttt{max}|\boldsymbol{\theta}-\hat{\boldsymbol{\theta}}|$. However, we introduce a complementary metric, adapted from the notion of success at the DNN level proposed in [CJM20], quantifying the error on the estimated pre-activation A_η value of the stolen neuron:

Definition 7.1 ($(\epsilon,\delta)-$successful extraction of a neuron) *A neuron η has been (ϵ,δ)-*

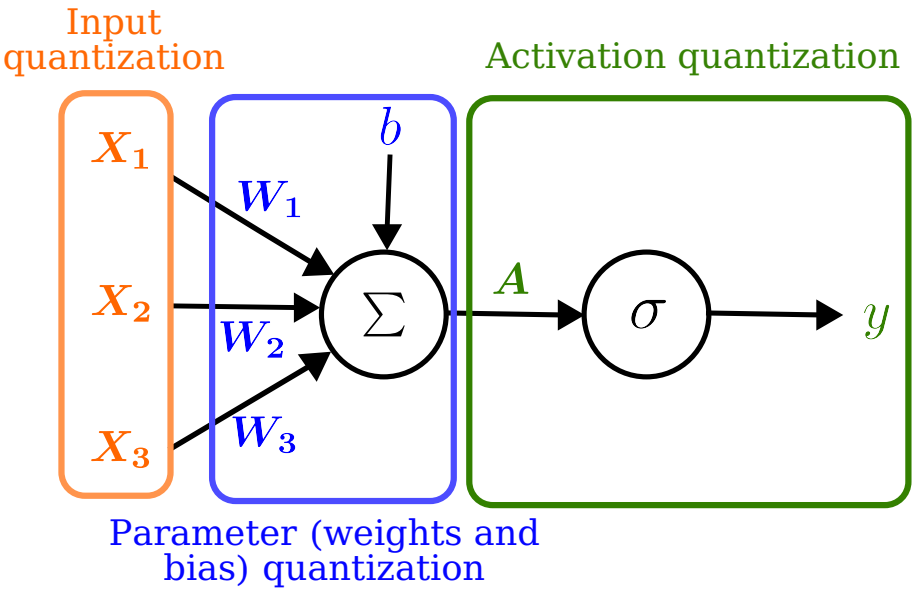


Figure 7.1 – Different targets for quantization in our experiments.

successfully extracted if

$$\mathbb{Pr}[|A_\eta - \hat{A}_\eta| \leq \epsilon] \geq 1 - \delta, A_\eta, \in \mathbb{R} \tag{7.1}$$

with A_η (resp. $\hat{A}_\eta$) the true (resp. estimated) pre-activation value of the targeted neuron η.

The idea behind this metric is that since the activation values of the targeted layer correspond to the input value of the next layer, this metric quantifies the viability of the attack at the next layer. Indeed, a high error in the activation values will lead to a very noisy system of equations for the next layer, resulting in an incorrect estimation of the weights for that layer. We measure this percentage for different values of ϵ but with a fixed value of 0.1 for δ and present the results in Figure 7.2. We can clearly observe the presence of a threshold below which the extraction's performance drastically falls. This suggests that for a low level of full quantization, *i.e.*, weights, biases, activation, and input quantization, of the targeted DNN, cryptanalytic extraction can no longer be performed. Indeed, the error in the targeted layer will cause an even higher error in the next layer, compromising the complete extraction of the DNN. Since we are not using the gradient to perform cryptanalytical extraction, this suggests that there is a different source of error in the extraction process occurring at low quantization levels that needs to be taken into

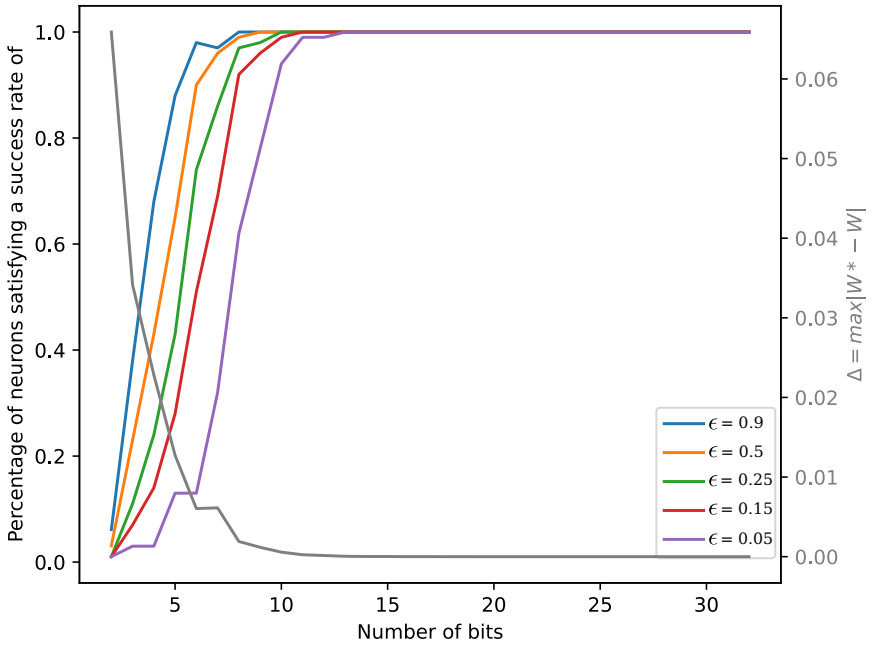


Figure 7.2 – Simulation of the impact of full quantization on the extraction of a perceptron for different quantization levels.

account to perform cryptanalytical extraction of quantized DNNs.

7.1.2 Impact of the Input Quantization

Cryptanalytic extraction is based on solving a system of equations formed by the critical points. This drop in performance must imply that there is a connection between the quantization of the targeted DNN and a noisier estimation of the critical points. Indeed, a wrong estimation of the critical points will lead to a biased equation, which will degrade the global system and therefore the estimation of the weights. To validate this intuition, we measure two quantities as a function of the number of bits used for quantization. Firstly, we measure the activation value of the perceptron for each extracted critical point. Since this value is supposed to be null (see Definition 4.7), it directly gives an indication of the bias in the system of equations. Therefore, we refer to this metric as *Error activation* noted Δ_{Act} in our experiments. Here, we assume that the extraction is performed via binary search, as in [Jag+20], and similarly to the methodology proposed in Chapter 6. As a consequence, it is only possible to guarantee that a point will be within

a certain distance of the hyperplane, and even without quantization, the activation values of the critical points are not null but close to zero. However, here we suspect that the distance of the critical points to the neuron-induced hyperplane is far larger than for non-quantized models, which causes the metric *Error activation* to increase with quantization.

To validate this hypothesis, we measure this distance to the hyperplane as a second metric, referred to as d_{HP} in our experiments. Since this metric is independent of the weights of the targeted neuron, it is complementary to the first one. To further identify the cause of the threshold, we use three sets constructed similarly to the one used in the previous section. The difference between these three sets is the quantization target. The first one corresponds to the set previously used with a full quantization, *i.e.*, input, weights, and bias. For the second one, the quantization target is the input. And finally, the last one corresponds to the quantization of the parameters, *i.e.*, weights and bias. We regroup the results in Figure 7.3.

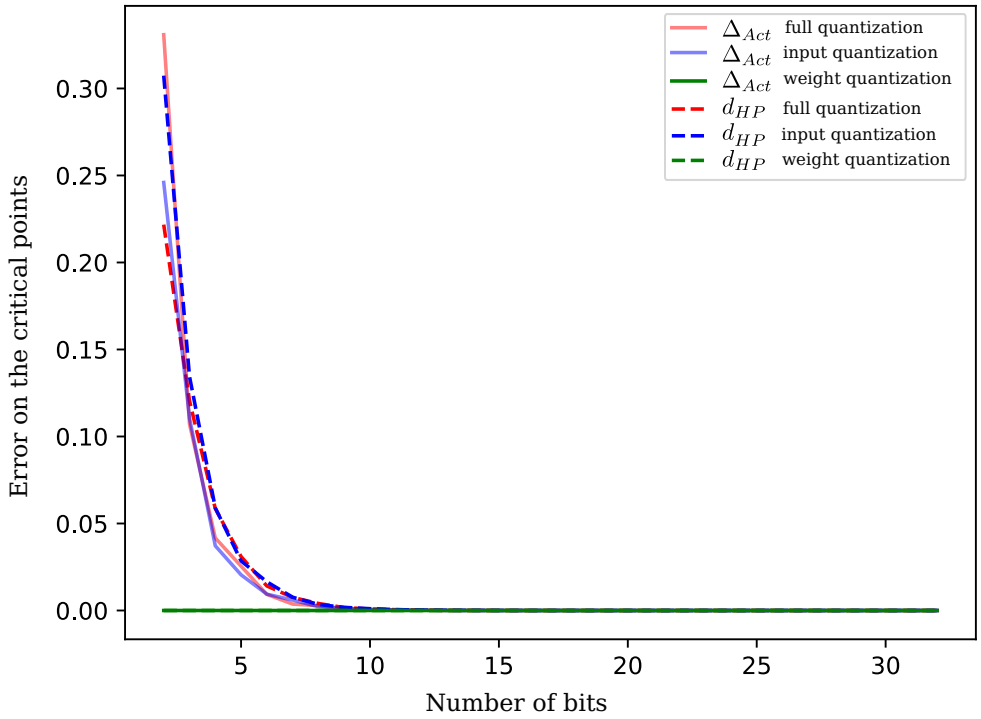


Figure 7.3 – Simulation of the estimation of the critical points for different quantization levels.

In the case where the quantization target is only the weights of the perceptrons, it

seems that no error is induced in the estimation of the critical point. This was expected as weight quantization only impacts the possible range of the weight values during the training process. During inference, the weights of the perceptrons correspond to a set of fixed real values independently of the number of bits used to encode their value, and therefore, quantization of just the weights has no direct impact on the extraction of the critical points. However, when the quantization target is either just the input or full quantization, *i.e.*, weights, bias, and input, we notice an inverse pattern in the evolution of the errors to the one observed in Figure 7.2, with a similar threshold around 10 bits. Our interpretation is that the error in estimating the critical points is caused by the quantization of the input. Indeed, input quantization reduces the number of bits in the representation of the inputs, which inherently reduces the range of potential input values.

We visually represent the impact of this reduction of the input space on the extraction of a neuron with two inputs, X_1 and X_2, in Figure 7.4. In the figure, the grid represents

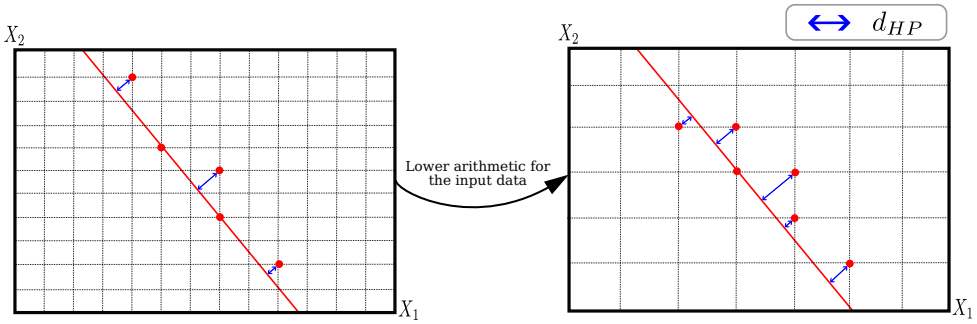


Figure 7.4 – Schematic representation of the impact of input quantization on the search for critical points.

the potential accessible values for a given quantization range. The background represents the input space here, $\mathbb{R}^2$, which would be accessible without quantization. Any point that is not on the grid will be rounded to the closest grid point before going through the linear layer of the perceptron. The mesh size of the grid directly depends on the number of bits of the quantization process. A lower number of bits will result in fewer possible values for X_1 and X_2, leading to a larger mesh size. In practice, the maximum distance between the grid and the hyperlane is bounded by the size of the unit square of the grid, which depends on both the number of bits used for quantization and the range of the data to be quantized. The issue for cryptanalytical extraction is that a larger mesh size will reduce the number of intersection points between the hyperplane and the grid. Therefore, the number of potential unbiased equations decreases with the number of bits, and the

error on the estimation of the critical points, *i.e.*, in the system of equations, increases. This is consistent with both the results from the simulation shown in Figure 7.2 and Figure 7.3. Furthermore, while the distance to the hyperplane is the cause, the true issue is the activation value of the biased critical points, which is also impacted by the input size. We test this hypothesis on a different set of perceptrons with varying input size for a fixed quantization step. All the perceptrons are quantized to 8 bits, and we test the three same quantization targets as previously. Finally, we use the same metrics as in Figure 7.3, *i.e.*, the distance to the hyperplane and the activation value of the critical points.

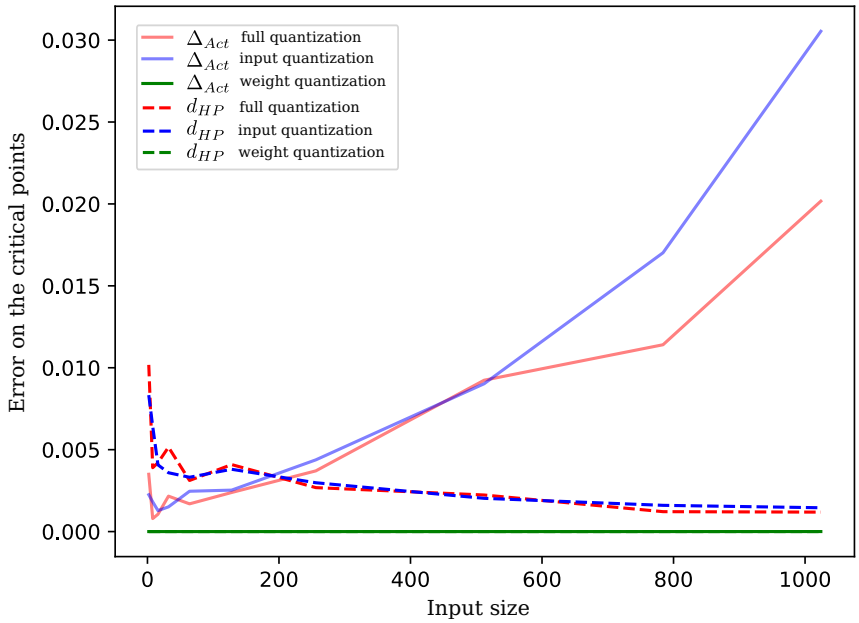


Figure 7.5 – Simulation of the estimation of the critical points for different input size with quantization at 8-bits.

We observe an increase in the averaged absolute activation value while the distance to the hyperplane remains stable as the input size increases. This was expected as the activation value corresponds to an accumulated sum. As such, increasing the size of the input vector augments the occurrence of critical points with biased components associated with large weights. In consequence, the performance of cryptanalytical extraction also decreases with the dimension of the quantized input vector. This result is confirmed by the evolution of the maximum error on the weights in the same settings. We only plot the results of this experiment for full quantization in Figure 7.6, as we expect the same behavior as for the previous experiments for the other quantization target, *i.e.*, similar

for input quantization and non-impacted for weights and bias.

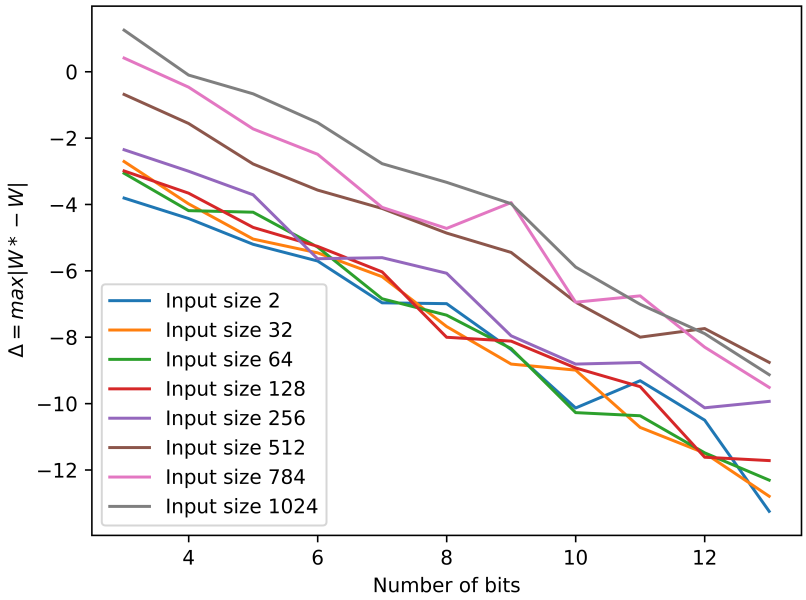


Figure 7.6 – Impact of the input size on the extraction of the weights of 8-bit fully quantized perceptrons.

Input quantization complicates the extraction of critical points by reducing the range of accessible values, thereby reducing the accuracy of the extracted weights. A potential solution would be to sample for a greater number of critical points and only keep those close to the grid in order to reduce their bias, but this would drastically increase the already large number of queries required for cryptanalytic extraction (see Subsection 6.3.4). Furthermore, such a criterion would be difficult to implement in practice, as no information on the distance of the critical points to the hyperplane is available to an attacker. Therefore, we propose introducing a new method based solely on side-channel analysis, enabling fidelity-based model extraction attacks robust to quantization.

7.2 Active learning with a fidelity objective

7.2.1 Moving from equalities to inequalities

As stated in the previous section, the issue with cryptanalytical extraction is that it relies on a set of critical equations constructed from the extracted critical points, whose estimations become more inaccurate with the quantization of the inputs. This introduces

a bias in the equations, which makes the system flawed and the solution imprecise. A straightforward solution is to account for this unknown bias in the system and transition from a system of equations to a system of inequalities.

Suppose, similarly to what has been done in Subsection 6.2.1, that the attacker has access to an oracle function $\mathsf{O} : \mathbb{N} \times \mathbb{N} \times \mathcal{X} \times \mathcal{X} \to \mathcal{B}$ that returns whether two inputs correspond to the same state for two given neurons. If we use this oracle function on a perceptron, denoted η, an attacker can collect two sets of inputs $\mathcal{X}_1$ and $\mathcal{X}_2$, such that the following three properties are respected:

$$\begin{aligned} &\forall (\boldsymbol{X}, \boldsymbol{Y}) \in \mathcal{X}_1^2, V(\eta; \boldsymbol{X}) = V(\eta; \boldsymbol{Y}) \\ &\forall (\boldsymbol{X}, \boldsymbol{Y}) \in \mathcal{X}_2^2, V(\eta; \boldsymbol{X}) = V(\eta; \boldsymbol{Y}) \\ &\forall (\boldsymbol{X}, \boldsymbol{Y}) \in \mathcal{X}_1 \times \mathcal{X}_2, V(\eta; \boldsymbol{X}) \neq V(\eta; \boldsymbol{Y}) \end{aligned} \tag{7.2}$$

Once these two sets are constructed, an attacker can make a hypothesis about which set corresponds to the active state and which one corresponds to the inactive state. Under this assumption, an attacker can, for any input $\boldsymbol{X} \in \mathbb{R}^n$ of the perceptron η described by its weights' vector $\boldsymbol{\theta}_\eta \in \mathbb{R}^n$ and bias $\beta_\eta \in \mathbb{R}$, have the following inequality:

$$\begin{aligned} &\sum_{i=1}^{n} \boldsymbol{\theta}_{\eta,i} \boldsymbol{X}_i + \beta_\eta \geq 0, \text{if } \boldsymbol{X} \in \mathcal{X}_1 \\ &\sum_{i=1}^{n} \boldsymbol{\theta}_{\eta,i} \boldsymbol{X}_i + \beta_\eta < 0, \text{otherwise} \end{aligned} \tag{7.3}$$

with the assumption that $\mathcal{X}_1$ (resp. $\mathcal{X}_2$) corresponds to the active (resp. inactive) state. By definition, we know that the optimal separation between the two classes corresponds to the neuron-induced hyperplane as described in Figure 7.7. It is possible to regroup both sets $\mathcal{X}_1$ and $\mathcal{X}_2$ into a global set $\mathcal{X}$. Since we know their state, we can also construct a set $\mathcal{V}$ corresponding to their state, *i.e.*, an input from $\mathcal{X}$ is associated with a state in $\mathcal{V}$, with the state being either active or inactive, depending on whether the input originates from $\mathcal{X}_1$ or $\mathcal{X}_2$. We can now use supervised learning on a dataset $D_{X,V}$ constructed from the sets $\mathcal{X}$ and $\mathcal{V}$ to find the hyperplane without relying on critical equations, using the states as labels.

While this method is straightforward for perceptrons, in DNNs, the high number of neurons requires an adaptation. In particular, while each neuron might converge toward

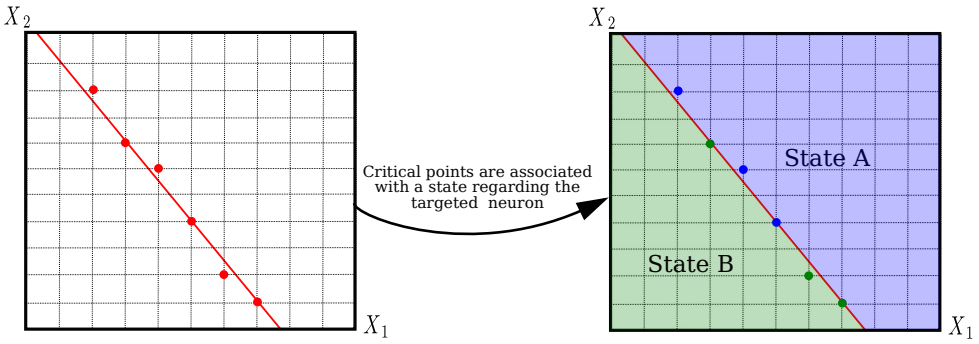


Figure 7.7 – Schematic representation of moving from critical equations to critical inequalities.

its true hyperplane if the states of each neuron are not consistent from one to another, the internal representation of the DNN will be different, reducing fidelity. Therefore, we employ the same method introduced in Subsection 6.2.3, and we align the sign of each neuron with that of a specific one. That way, all hyperplanes are aligned, and an attacker has only to train two models corresponding to the two hypotheses for this neuron.

Similarly to what has been done in Chapter 6, we will use a side-channel attack to compare the states of neurons for two different inputs. We will further describe the side-channel attack that we use as an oracle function in Subsection 7.3.2. In the next section, we outline our comprehensive methodology based on these principles and describe the algorithm that we will use in this chapter for quantized DNNs.

7.2.2 Global Algorithm

In this section, we describe our algorithm to target a model $f_{\boldsymbol{\theta}}$ composed of L layers. Similarly to Chapter 6, we suppose to have access to the architecture of $f_{\boldsymbol{\theta}}$, and as such we randomly initialize a model g with the same architecture. Furthermore, we suppose access to a similar oracle function based on a side-channel attack. We recall that we use the notation g_l to represent the composition of affine transformations and activation functions from the first layer to layer l (see Subsection 1.1.3).

Our methodology corresponds to a two-step process. A first step is used to mimic the internal representation of the targeted model. In the second step, we train the substitute model on the task of the targeted model. In this section, we first describe these two steps and then introduce our global algorithm.

First step: Approximation of the hyperplanes. The first step of our methodology is illustrated in Figure 7.8. We first collect a large number of side-channel traces associated

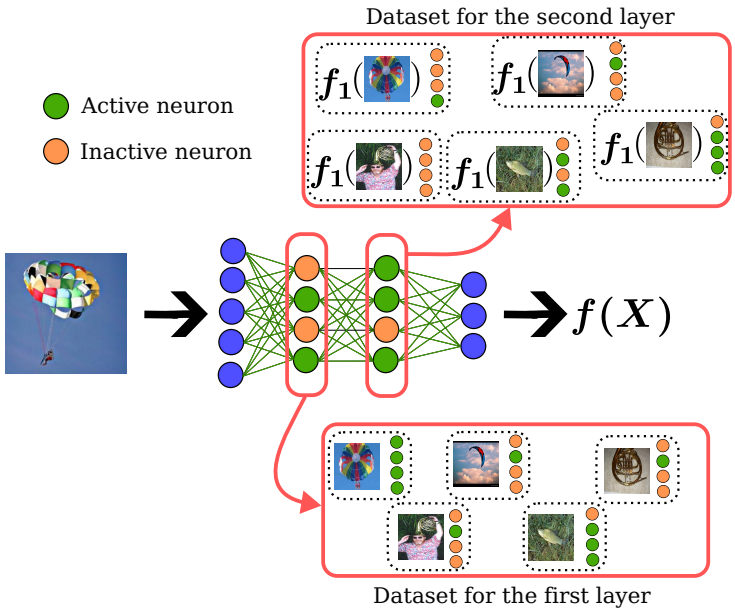


Figure 7.8 – Schematic representation of our first step on an MLP with two hidden layers.

with the processing of a set of m input tensors $\mathcal{X} = \{\boldsymbol{X^1}, \boldsymbol{X^2}, ..., \boldsymbol{X^m}\}$. We then use our oracle function to compare the states of each input, for each neuron in each layer. Then, following the method introduced in Subsection 6.2.3, we align each group on one neuron. Once this step is performed, we know the state of all neurons relative to that one specific neuron η. We then make a hypothesis on the state of neuron η, which enables the attacker to know the state of all neurons, and construct a set of tensors $\mathcal{V}_i = \{\boldsymbol{V_i^1}, \boldsymbol{V_i^2}, ..., \boldsymbol{V_i^{L-1}}\}$, with each tensor $\boldsymbol{V_i^l}$ corresponding to the states associated with $\boldsymbol{X^i}$ for the neurons in layer l under the hypothesis made on the state. We do not include the final layer L, as this one does not include an activation function, and as such, we do not have any information on the state of the neurons for this layer. We construct this set for all m input tensors and form a global set $\mathcal{V}$ corresponding to the set of all the states associated with all inputs from $\mathcal{X}$ for all neurons.

We then perform supervised learning on each layer sequentially, starting with the input layer. We create a dataset $D_{\boldsymbol{X_l},\boldsymbol{V_l}}$ at each layer corresponding to the input set after passing through the first $l-1$ layers, and the set of states associated with layer l. For the first layer, no transformation is needed, and $D_{\boldsymbol{X_l},\boldsymbol{V_l}}$ can be directly obtained from the set of

input tensor $\mathcal{X}$ and from $\mathcal{V}$. We then compute the loss between the estimated activation values of the lth layer, $\hat{\boldsymbol{A}}_l$, and the expected state stored in $\boldsymbol{V_l}$. Since this is very close to a binary classification problem, we use binary cross-entropy as our loss function. We only update the weights associated with the current layer of g.

Once the training is finished for the lth layer, we move to layer $l+1$. We compute the new dataset $D_{\boldsymbol{X_{l+1}},\boldsymbol{V_{l+1}}}$ by applying g_l on $\boldsymbol{X_l}$ and updating $\boldsymbol{V_{l+1}}$ with $\mathcal{V}$. We then repeat the same training process for each layer, except for the final one, *i.e.*, layer L. Indeed, for this one, there are no activation functions and, as such, no states to guide our training process. Therefore, this layer is initially randomly initialized and not trained. The training of this layer is performed in the second and final step of our methodology.

Second step: Extraction of the last layer and fine tuning the complete model. At this point in our algorithm, the first $L-1$ layers of g are trained, and the layer L remains in its random initial state. The information that we have to train this model on the task is only the hard-labels from the targeted model. We use these with the original input set $\mathcal{X}$ to train the substitute model with a classical cross-entropy loss. However, contrary to the previous step, this time all the weights are updated via backpropagation. To maintain the interest of the first step, we add a penalty to the classical cross-entropy loss corresponding to the loss on the state summed over all layers. We refer to the loss used to train the substitute model, composed of the classical cross-entropy loss and the penalty, as $\texttt{loss}_{full}$.

Complete algorithm. We now provide a full description of our complete attack in Algorithm 4. To ease the readability of the algorithm, we make two simplifications. The first one is that the input set $\mathcal{X}$ is restricted to only one batch of input vectors $\boldsymbol{X}$. The second point is that the training algorithm is applied only for one epoch in both steps 1 and 2. Both of these simplifications can be easily implemented using *for*-loops.

During the first part of our attack, we utilize side-channel information to infer the sign of each activation value through our oracle function. This is straightforward for linear layers where one activation corresponds to one neuron. However, in the case of convolutional layers, the filter is applied sequentially to the input feature map, producing an output feature map on which the activation function will be applied (see Subsection 2.1.1). Therefore, it is possible in this case to generate more than one inequality per input for such layers.

Algorithm 4 Extraction of quantized model via supervised learning.

Require: The targeted model $f_{\boldsymbol{\theta}}$ composed of L layers, a batch of input vectors $\boldsymbol{X}$, O_l an oracle function returning all the states of the neuron in layer l, a training algorithm such as gradient descent which can be applied layer-wise A_l or on the full model A, and g a empty substitute model with the same architecture as $f_{\boldsymbol{\theta}}$.

Ensure: Perform a fidelity-based model extraction of $f_{\boldsymbol{\theta}}$

$\boldsymbol{Y} \leftarrow \texttt{argmax}(f_{\boldsymbol{\theta}}(\boldsymbol{X}))$ ▷ Extraction of the hard-labels

for $l \leftarrow 1$ to L **do** ▷ Extracting the neurons'state for each layer

  $\boldsymbol{V_l} \leftarrow \mathsf{O}_l(f_{\boldsymbol{\theta}}(\boldsymbol{X}))$

end for

for $l \leftarrow 1$ to $L-1$ **do** ▷ Step 1

  $\hat{\boldsymbol{A}}_l \leftarrow g_l(\boldsymbol{X})$ ▷ Activation values of layer l

  $\texttt{loss}_{side} \leftarrow \texttt{loss}_{BCE}(\hat{\boldsymbol{A}}_l, \boldsymbol{V_l})$ ▷ Loss of the current layer

  $g_l \leftarrow \mathsf{A}_l(g_l, \texttt{loss}_{side})$ ▷ Updating only the weight of layer l

end for

$\texttt{loss}_{side} \leftarrow 0$

for $l \leftarrow 1$ to $L-1$ **do** ▷ Step 2

  $\hat{\boldsymbol{A}}_l \leftarrow g_l(\boldsymbol{X})$

  $\texttt{loss}_{side} \leftarrow \texttt{loss}_{side} + \texttt{loss}_{BCE}(\hat{\boldsymbol{A}}_l, \boldsymbol{V_l})$

end for

$\texttt{loss}_{task} \leftarrow \texttt{loss}_{CE}(g(\boldsymbol{X}), \boldsymbol{Y})$ ▷ Loss on the task

$\texttt{loss}_{full} \leftarrow \texttt{loss}_{task} + \texttt{loss}_{side}$

$g \leftarrow \mathsf{A}(g, \texttt{loss}_{full})$ ▷ Training on complete model

return g

The intuition behind our methodology is that it is based on two distinct loss functions. The first one corresponds to the classical task loss, which is used in most active learning scenarios. This is the loss function computed for the last layer with the hard-label. We refer to this loss as $\texttt{loss}_{task}$. In our experiment, it corresponds to the cross-entropy loss $\texttt{loss}_{CE}$ between the hard-labels from the targeted model and the output of our stolen model during training.

The second loss corresponds to the loss function associated with the difference of states between the targeted and extracted neurons. Its purpose is to augment the previous loss with supplementary information on the internal representation of the targeted model obtained with our side-channel-based oracle. We refer to this second loss as $\texttt{loss}_{side}$, and in our experiment, we use the binary cross-entropy loss $\texttt{loss}_{BCE}$ (see Definition 1.8).

While both losses aim for fidelity, they operate on different parts of the model. The

first loss considers the outputs of the model and is minimized when maximum agreement between the hard-labels of the targeted and stolen models is achieved. The second loss tries to align the neuron-induced hyperplanes from both models to minimize the difference between their internal representations. If the second loss, $\texttt{loss}_{side}$, is null for all inputs, then Equation 7.3 is respected for all inputs and for all neurons, which implies that the hyperplanes from the targeted and the stolen model are the same. In this case, their signature are also the same (see Subsection 4.3.4), and the two models are functionally equivalent (see Definition 4.4). A more physical interpretation of the second loss function is to view it as a loss function that attempts to replicate the side-channel behavior of the targeted model in the stolen model. Indeed, if the hyperplanes are aligned, then the states are the same for all inputs, and as such, the processing associated with the ReLU function is the same, and the deterministic part of the EM emanations (see Equation 3.1) is the same between the targeted and the stolen model.

While our methodology is presented independently of the input, we expect that the performance of the supervised learning approach will be greatly impacted by the set of inputs $\mathcal{X}$ used. Therefore, in the following section, we further describe our choice for this set.

7.2.3 Critical vs non-critical inequalities

One of the most straightforward choices for a dataset is to use Algorithm 2, which performs binary search and utilizes the extracted biased critical points as inequalities. Such inequalities are referred to as critical inequalities. Indeed, those points are, by definition, as close as possible to the hyperplanes and therefore encapsulate more information on the hyperplanes compared to random points. Such an approach is similar to the work presented in [BM24], where the authors search for samples with low confidence, *i.e.*, close to the decision boundary of the model, to train their substitute model.

While this approach is effective, we argue that it does not fully leverage the potential of the system based on inequalities. Indeed, at low quantization levels, the extracted model will be an approximation of the targeted model, independently of the choice of dataset used. Furthermore, while the search for critical inequalities leads to a dataset composed of high-information points, it is also very query-intensive and dependent on the quantization. By definition, the binary search is a query-intensive process. However, the major issue is that this process needs to be performed for all of the neurons sequentially. In this scenario, the attacker would use different sets of inputs at each layer during step 1

of our attack. Finally, the quality of critical inequalities, *i.e.*, the distance of the points to the hyperplane, highly depends on the input quantization as described in Subsection 7.1.2.

Therefore, we argue that we need to propose a different methodology for fidelity-based model extraction, independent of the input quantization and less query-intensive. With that objective in mind, we use a common dataset for the extraction of all layers in the rest of this chapter. Indeed, in such a scenario, the attacker can completely subdivide the attack into two steps. The first step corresponds to extracting the state of the neurons for the input $\boldsymbol{X}$ using the oracle function O. Then the second, which can be performed remotely from the targeted device, uses Algorithm 4 to train a substitute model with no adaptation to the algorithm. This methodology is free from any assumption on the quantization, only requires a side-channel attack, and is far more query-efficient since it does not perform a critical point search at each neuron. However, the points used can be far from the hyperplane, and as such, the training process is less precise compared to using critical inequalities.

While this choice reduces the fidelity of the substitute model, we argue that it corresponds to a different type of fidelity-based model extraction compared to cryptanalytical methods. In the next section, we further elaborate on this point and introduce different levels and objectives within fidelity-based model extraction attacks.

7.2.4 The Different Levels of Fidelity-Based Model Extraction Attacks

In [Jag+20], Jagielski *et al.* introduce the taxonomy related to model extraction and the difference between fidelity-based model extraction attacks and accuracy-based model extraction attacks. However, due to the complexity of fidelity-based model extraction attacks, several methodologies have emerged with different evaluation metrics and even objectives.

In this thesis, we primarily consider cryptanalytical-based approaches, such as those described in [Jag+20; CJM20; Car+25; Che+25a]. Operating at the same level, we also include side-channel attacks directly extracting the weights through CEMA, such as in [Bat+19; Hor+25; Jou+23a]. On the other hand, attacks such as the one described in [BM24] utilize low-confidence samples to retrain a DNN, without consideration for the architecture, to steal a DNN as close to the targeted one. This type of attack can lead to very different internal representations due to the high level of the methodology, *i.e.*,

supervised learning on the output of the model. And, in between these two broad types, we consider attacks that perform training of a substitute model under constraints learned from the targeted model. This group includes previously presented fault-based attacks [Hec+23; Rak+21] and the methodology presented in this chapter.

We argue that these attacks operate at different levels, and while they all correspond to fidelity-based model extraction, they do not share the same objective in terms of fidelity and therefore should not be evaluated using the same metrics. Indeed, cryptanalytical extraction and side-channel attacks operate at the weight level and try to extract a stolen model with stolen weights as close as possible to those of the targeted model. As such, the most natural metric to evaluate these attacks is the maximum absolute difference between the extracted and the stolen weights. We argue that the other metrics, *e.g*, fidelity, AuA, do not truly represent the success of a cryptanalytical extraction. Indeed, due to the complexity of the targeted DNN, it is possible to achieve strong results using these metrics while having very different models. Our results presented in the previous chapter (see Subsection 6.3.4) validate this concern. While the extraction of the last layer leads to a significantly different model, we still achieve a strong AuA and fidelity. A similar concern can be made about the methodology proposed in [CS25]. Therefore, we make the point that methods based on cryptanalysis should be solely evaluated with the maximum error on the weights.

On the other hand, using this metric to evaluate the attack introduced in [BM24] is not relevant, as the purpose of this attack is to extract a model with the same decision boundary. Therefore, extracting a model that predicts exactly the same class for all inputs, but with a completely different set of weights, would still be considered a successful extraction. Similarly, the extraction of DNNs using supervised learning under constraints does not provide guarantees that the model will share the same parameters, but rather aims to increase the similarity between the internal representations of the stolen and targeted models. For these objectives, we argue that it is more natural to evaluate the fidelity and the AuA of the stolen model.

7.3 Practical Experiments

7.3.1 Experimental Setup and Threat Model

Threat model. In this work, we consider an attacker whose purpose is to perform fidelity-based model extraction of the targeted DNN. The attacker operates in hard-label settings, *i.e.*, the output of the targeted model is only composed of the output class and not of the output probabilities. The attacker is supposed to have access to a substitute model with the same architecture, *i.e.*, the architecture is known, and we suppose access to a subset of the training dataset. We set the size of this subset to 10% of the total dataset, which is common in the literature [Rak+21; Hec+23]. We also suppose that the substitute and the targeted model use the same data format and that the input quantization is known. This corresponds to a specific scenario where the quantization is not performed by an AI framework but directly on the inputs before sending them for inference, as in [Ma20]. Finally, we consider that the attacker is able to send inputs to the targeted model and has physical access to the device, which allows for the acquisition of EM traces during the processing of these inputs. However, we do not require any clone device to perform a profiling phase, as we use a non-profiled side-channel attack. This corresponds to the same threat model as the one described in Chapter 6, except for access to the input quantization and a subset of the training dataset. Therefore, we also consider this attack to be a non-profiled grey-box attack.

Targeted architecture and datasets. We first validate our methodology on two simple architectures, an MLP composed of two hidden layers trained on the MNIST dataset and one convolutional neural network with three convolutional layers using 5 by 5 kernels, each followed by a pooling layer. We provide the full description of these architectures, respectively, in Table 7.1 and Table 7.2.

Table 7.1 – Description of the architecture of the MLP.

Layers	**Output size**	**Parameters**
Flatten	1×784	0
Dense + ReLU	1×128	$100,352$
Dense + ReLU	1×64	$8,192$
Dense + ReLU	1×10	640

We then further demonstrate the potential of our methodology by targeting several

Table 7.2 – Description of the architecture of the CNN.

Layers	**Size**	**Parameters**
Conv2D + ReLU Average pooling	$32 \times 5 \times 5$ 2×2	$2,400$
Conv2D + ReLU Average pooling	$32 \times 5 \times 5$ 2×2	$25,600$
Conv2D + ReLU Average pooling	$64 \times 5 \times 5$ 2×2	$51,200$
Flatten		
Dense + ReLU	-	$10,240$

classifiers designed using SOTA architectures. We especially perform fidelity-based model extraction of VGG11, ResNet18, and ResNet34, all trained on the CIFAR-10 dataset.

Experimental Setup. Similarly to what has been done in Chapter 5, we target quantized DNNs embedded on an ARM Cortex-M7 MCU on an STM32-F767 board. We also use NNOM [Ma20] for the experiment in this chapter, as it is an efficient and open-source framework. We compile the project without compiler optimization. Finally, since the original implementation of the ReLU function in NNOM is not constant-time, it can be targeted to extract the weights or inputs as demonstrated in [MBC21]. Therefore, we modify the source code of the function `local_relu_q7()` to make it constant time using the implementation proposed as a side-channel countermeasure in [MBC21].

We use a probe from Langer (EMV-Technik RF-U 2.5, with a frequency range of 30 MHz to 3 GHz) as well as an amplifier (2X60-3018G, with a 12V alimentation) to capture the EM side-channel leakages using a Lecroy oscilloscope (2.5 GHz WaveRunner 625Zi). To ease the leakage exploitation, we modify the targeted function (*i.e.*, `local_relu_q7()` from NNOM) and add a trigger surrounding its execution. The configuration of the trigger is not considered difficult, as the pattern associated with the processing of the ReLU is different from the one associated with the surrounding matrix multiplication.

7.3.2 Analysis of the Side-Channel Vulnerability

Following the experiments performed in Chapter 5, we use the NNOM framework to validate our methodology. In the experiment previously performed (see Section 6.3), X-Cube-AI used a constant-time implementation of the ReLU function. However, as mentioned in Subsection 7.3.1, this is not the case for the NNOM framework. Therefore,

we modify the function to make it constant time, following the recommendation from [MBC21] to place ourselves in a more realistic scenario. We provide the pseudo-code of our implementation in Algorithm 5.

Algorithm 5 Pseudo-code of the constant-time ReLU function.

Require: Pre-activation value $z \in \mathbb{R}$
Ensure: $a = \texttt{max}(0, z) \in \mathbb{R}^+$
$a \leftarrow (\sim (z \gg 7))\ \&\ z$

This implementation was already described in Subsection 6.3.3, and this pseudo-code corresponds only to an adaptation of the algorithm. However, we include the assembly code associated with this implementation of the ReLU function in Figure 7.9. As expected,

```
1  ; data[i] = (~(data[i] >>7)) & data[i];
2  ldr r2, [r7, #4]     ; Loading the address of the activation vector
3  ldr r3, [r7, #12]    ; Loading the address of the index "i"
4  add r3, r2           ; Set the pointer to the i-th element of
       activation vector
5  ldrsb.w r3, [r3]     ; Loading of the i-th element of z in r3
6  asrs  r3, r3, #7   ; Bitwise shift of r3 give the bit sign
7  sxtb  r3, r3       ;
8  mvns  r3, r3       ; Bitwise Not of the bit sign
9  sxtb  r1, r3       ; Mask is stored in r1
10 ldr r2, [r7, #4]     ; Loading the address of the activation vector
11 ldr r3, [r7, #12]    ; Loading the address of the index "i"
12 add r3, r2           ; Set the pointer to the i-th element of
       activation vector
13 ldrsb.w r2, [r3]     ; Loading of the i-th element of z in r2
14 ldr r0, [r7, #4]     ;
15 ldr r3, [r7, #12]    ;
16 add r3, r0           ; r3 points to Z[i]
17 ands  r2, r1       ; Bitwise And between the mask r1 and the true
       value r2
18 sxtb  r2, r2       ;
19 strb  r2, [r3, #0]     ; Store the result in Z[i]
```

Figure 7.9 – Assembly code from the NNOM ReLU function.

we observe that the bit sign is extracted and used to create a mask (see lines $1-9$). During the load instruction, leakages correlated to the sign, and therefore to the state of the neuron, should be observed. Furthermore, in line 17 of Figure 7.9, the application of the mask will either cause transitions $1 \rightarrow 0$ for the 8 bits or leave the value unchanged. This difference in processing should be visible in the amplitude of the observed signal.

To validate our code analysis, we compute the SNR associated with the processing of the ReLU function for one layer of 64 neurons. A high SNR value indicates a strong correlation between the observed EM emanations and the state of one neuron. To improve readability, we only display the SNR values associated with the first ten neurons in Figure 7.10.

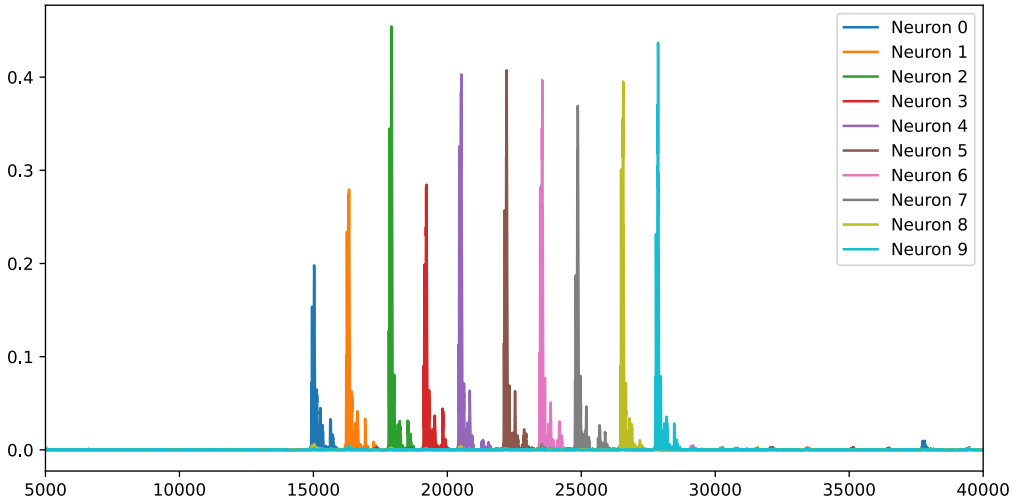


Figure 7.10 – SNR associated with the first 10 neurons in one layer of 64 neurons.

This plot is obtained with $20,000$ non-resynchronized traces in the experimental set-up described in Subsection 7.3.1. As expected, we observe a strong correlation between the EM traces and the states of each neuron sequentially. We then use the same methods presented in Chapter 6 to extract the state of the neuron from the EM traces. We achieve a success rate of 89.98% in one EM trace on all neurons (the results are averaged with 1994 traces for one neuron and then across all 64 neurons within the layer). As the implementation of the function remains the same throughout the DNN, we consider that we are able to target neurons using the ReLU function independently of their depth. Once we have validated that the side-channel vulnerability is also present in NNOM, we test our complete methodology.

7.3.3 Results with a Perfect Oracle Function

We validate our methodology by targeting all architectures presented in Subsection 7.3.1. As mentioned, we suppose access to a subset of the training data, as in [Rak+21; Hec+23]. More specifically, we suppose access to $4,000$ images in both the MNIST and CIFAR-10 datasets. We first present our results on the two simple architectures described in Subsection 7.3.1, *i.e.*, the MLP and the CNN, in Table 7.3 to validate our methodology. Then, we test our extraction methodology against SOTA architectures and summarize the results in Table 7.4. We compute three metrics for comparison following the point made in Subsection 7.2.4: accuracy, fidelity, and AuA. Accuracy and fidelity are averaged on the complete test set of the true task, *i.e.*, either MNIST or CIFAR-10, and AuA is evaluated using the same methodology as in Chapter 6. Finally, we average each metric on a pool of ten targets for each architecture.

To provide a comparison in Table 7.4, we define the *baseline* methodology with a simple active-learning extraction, where a model is trained in the same settings, *i.e.*, same architecture, number of data, same hyperparameters, *etc.*, as the ones used in our experiments. The only difference between the baseline methodology and ours is the additional information given by the side-channel-based oracle. Finally, we also give the results in white-box settings. In this scenario, the accuracy is the same as the target, and the AuA is 0%, as all adversarial examples are directly generated on the target model. This scenario corresponds to the ideal scenario and is used, as such, as a limit.

Table 7.3 – Evaluation of our methodology on simple architectures.

Method	**MLP**			**CNN**		
	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)
Baseline	93.47	93.73	85.19	50.48	52.86	40.58
Side-channel-based oracle	97.37	98.34	48.38	65.39	74.86	25.02
White Box	97.65	100	0	68.36	100	0

On the simple architectures, our methodology outperforms the baseline methodology on all three metrics, *i.e.*, accuracy, fidelity, and AuA, with an increase of 4.61% (resp. 22%) in fidelity on the MLP (resp. CNN). We hypothesize that there are two reasons for the significant difference in performance between our methodology and the baseline. The first one is due to the nature of the target. For the MLP, only one inequality per neuron is provided for each input. For the CNN, each kernel is moved across the input feature map, generating several activation values, *i.e.*, inequalities, at each step of the convolution.

The second reason is linked to the task of the targeted model. Fidelity is evaluated as label agreement, and as such, very different models giving the same prediction will have high fidelity. And since MNIST is a simple enough task for a DNN, the fidelity metric is almost equivalent to accuracy due to the low error rate. In consequence, even our *baseline* methodology can achieve high fidelity on such a dataset. However, this is no longer the case for more complicated datasets such as CIFAR-10, as we can observe in Table 7.3. In terms of AuA, we observe that despite having a very high fidelity, the transfer rate is very low for the MLP, both with the baseline methodology and our method. We hypothesize that the poor performance of both the baseline and our methodology is connected to our previous observation on the MNIST dataset. Indeed, due to the simplicity of the MNIST dataset, the high value of the fidelity metric does not guarantee that the extracted DNN truly behaves as the targeted DNN. And consequently, this translates to a low transfer rate, *i.e.*, high Accuracy under Attack, of adversarial examples between the two networks. For the CNN, our methodology improves the transfer rate by 15% compared to the baseline, indicating potential interest for an attacker attempting to mount further attacks.

Comparison with Cryptanalytical-Based Attacks. To provide comparison, we also evaluate cryptanalytical extraction following the methodology introduced in Chapter 6 on the quantized MLP. To match the original threat model, we suppose that no access to the training data is available. In these settings, we achieve an accuracy for the stolen model of 69.74% and a fidelity of 70.07% on the test set. Finally, the AuA of the targeted DNN is of 98.42%, which is even higher than the original accuracy. These results are directly related to the issues mentioned in Section 7.1. Indeed, for the first layer, the maximum error on the extracted weights is $\texttt{max}|\Delta W_{L1}| = 2^{-2.34}$ due to the wrong estimation of the critical points. This leads to incorrect equations for the second layer, and the error accumulates at each layer. For the second layer, we have $\texttt{max}|\Delta W_{L2}| = 2^{1.77}$, and this causes too many errors on the extraction of the final layer via supervised learning, which explains the limited performance on this example. This example highlights the limitations of cryptanalytical-based methods on quantized models.

Now that we have validated that our methodology is efficient on simple examples, we test our attack on SOTA architectures. In our experiments, we successfully extract all architectures with a high fidelity, *i.e.*, superior to 90%. We also improve the accuracy (resp. fidelity) by 20.4% (resp. 28.4%) on average across all architectures tested compared to the baseline method. Our evaluations show that our methodology is more driven by

Table 7.4 – Evaluation of our methodology on state-of-the-art architectures.

Method	ResNet18			ResNet34			VGG11		
	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)
Baseline	61.88	63.15	40.60	62.74	63.57	40.12	62.72	64.33	36.82
Side-channel-based oracle	83.57	93.61	8.87	83.68	92.49	9.95	81.34	90.16	10.92
White Box	83.25	100	0	83.77	100	0	82.40	100	0

a fidelity purpose than classical learning-based methods. Indeed, in Table 7.4, we can clearly observe that for the baseline method, the accuracy and fidelity remain close for all architectures, with an average absolute difference between the two metrics of 1.2%. On the other hand, with our method, the average absolute difference between these two metrics is 9.2%. This result stems from the fact that our methodology penalizes substitute models with hyperplanes, or equivalently, side-channel behavior, that is too different from the targeted model, which increases the fidelity of the stolen model.

Finally, in terms of AuA, we also observe that our methodology improves the transfer rate of adversarial examples on the targeted DNN by an average of 28.0%. As expected, the AuA for the baseline methodology is close to 40.0% on all architectures. On the other hand, we achieve close to white-box performance for the ResNet18, ResNet34, and VGG11 with an average AuA of 9.9%, demonstrating the validity of our methodology.

7.3.4 Considering Errors in the Oracle Function

In our previous experiments, we suppose that the extraction of the state is successful for all activation values for all layers. However, in the case of deep convolutional layers, this can be a very restrictive assumption, as the size and number of feature maps, *i.e.*, the number of activation values, grow rapidly. Indeed, for VGG11, this corresponds to performing the side-channel attack on $152,576$ values for one image. This large number is a strong assumption in our attack, as in Table 7.3 and in Table 7.4, the results are obtained under the assumption that the side-channel attack is always successful in one trace for all neurons. Furthermore, the number of side-channel attacks to perform is deeply connected to the input size, and as such, it would largely increase on a dataset such as ImageNet. To improve the feasibility of our attack, we investigate the impact of the performance of the side-channel attack on the fidelity and the accuracy of the extracted model. In Figure 7.11, we illustrate the evolution of the performance of our methodology on a VGG11 with respect to the percentage of wrongly estimated states. More specifically,

we plot the evolution of the accuracy and the fidelity of both the *baseline* method and our attack using the side-channel-based oracle.

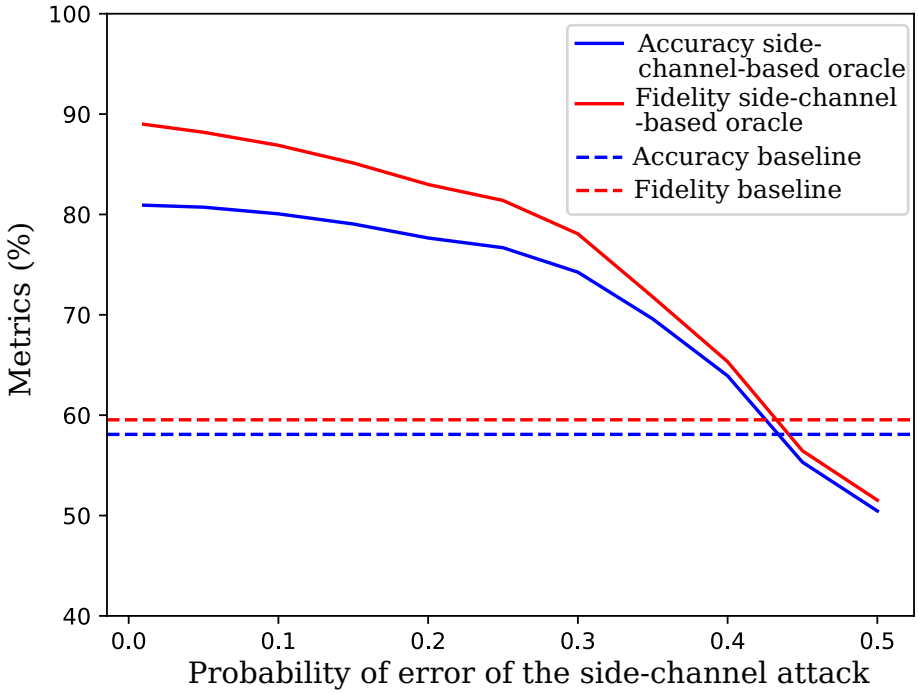


Figure 7.11 – Impact of errors in the side-channel attacks on our extraction methodology on the example of a VGG11 model.

In this scenario, we first extract all the signs of the activation values associated with our inputs with a certain probability of success. This leads to a dataset composed of the same set of inputs as in the previous experiments, but this time with incorrect state estimations. The maximum percentage of wrongly estimated states is set to 50%, which is equivalent to assigning the state randomly. We then follow the same methodology introduced in Section 7.2 to learn the hyperplane associated with the targeted model. We average this result on five different models with the same architecture and training hyperparameters to assess the impact of an imperfect side-channel-based oracle on the performance of the complete attack.

In Figure 7.11, we observe the robustness of our methodology to a degradation of the performance of the side-channel-based oracle, and even that an imperfect side-channel attack can improve the performance of the global learning process. Furthermore, we observe in Figure 7.11 that our methodology continues to outperform the baseline methodology on this model, with a probability of error of 0.4. We hypothesize that both these results

are due to the fact that the internal representations of large models are so complex that it is possible to achieve performance comparable to that obtained via naive active learning, despite having a completely different internal representation. This result further highlights the fact that naive active learning methods are not adapted in the context of fidelity as an objective.

Moreover, in Figure 7.11, we observe that even with a probability of 0.3 that a given state is incorrectly estimated for a neuron, the drop in fidelity is below 10% compared to a perfect attack. This result proves that not all activation values are necessary to guide the learning process and increase the fidelity compared to the baseline method. Therefore, we use this result to relax the threat model of our methodology. More specifically, we test the performance of our methodology on a very restricted scenario where only the sign of one activation value is correctly assigned for each kernel. The rest are not known. This is equivalent to performing our attack perfectly a single time for each kernel. We refer to this attack as a single side-channel attack, in opposition to the previous attack, which takes advantage of the complete knowledge of the state of the targeted model. In this scenario, the dataset constructed in the first step of our attack (see Subsection 7.2.2) contains far less information on the internal states compared to previously. On the other hand, this corresponds to a more suitable and efficient attack scenario due to the reduction in the number of required calls to the oracle function. We include the results obtained under this assumption in Table 7.5. No results are available for the MLP in this context. As this model does not include convolution, it already only has one state per neuron.

Table 7.5 – Evaluation of our methodology using a single activation value per neuron.

CNN			**ResNet18**			**ResNet34**			**VGG11**		
Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)
61.98	69.56	29.26	75.69	79.27	26.70	77.93	81.35	26.47	77.41	81.87	21.18

We demonstrate that, despite this restriction on the access of the side-channel-based oracle, we still outperform the baseline method (see Table 7.3 and Table 7.4) in terms of accuracy by an average of 14.9% and improve fidelity by 18.6% on average. However, the performance drops compared to the results obtained with full access to the internal states, with an average accuracy (resp. fidelity) drop of 4.1% (resp. 8.0%). As expected, due to the reduction in internal information provided by the oracle function, the fidelity is subject to a higher reduction compared to the accuracy. This experiment demonstrates

that our methodology remains efficient even in a more relaxed threat model, highlighting the trade-off between the number of side-channel attacks and the final fidelity of the stolen model.

7.4 Discussion and Perspectives

7.4.1 Comparison with Different Threat Models

Due to our threat model, our experiments are closely aligned with those presented in [Hec+23; Rak+21]. Indeed, both works assume physical access to the device on which the AI is deployed, as well as access to a subset of the training dataset and to the architecture. However, these works employ fault-based attacks to extract a subset of the bits of the DNN's weights and therefore suffer from restrictions related to this method. In [Hec+23], the authors use a profiling device to determine the precise mapping of the weights' bits in the external memory before the attack. Finally, Rakin *et al.* [Rak+21] use the RowHammer attack to fault the bit associated with the DNN's weights, which is restricted to DRAM memory.

While these works correspond to different threat models, the associated methodologies are also fidelity-based model extraction attacks. Moreover, they fall in the same category of objectives within fidelity that we introduced in Subsection 7.2.4. Therefore, we provide a comparison in this section with the results obtained from these methods. For both methods, we compare the results presented in [RHF19; Hec+23] with our complete and single-attack results. We begin by comparing the MLP and the CNN, which are already discussed in [Hec+23] in Table 7.6. We furthermore compare our methodology with the one described in [Rak+21] on ResNet-18, ResNet-34, and VGG11 in Table 7.7. Due to the costly nature of our side-channel-based attack, we choose to include the results from [Rak+21; Hec+23] with the largest number of MSB extracted with fault-injection, to improve the comparison with our complete attack.

In Table 7.6, we observe that we achieve stronger results on the MLP in terms of accuracy and fidelity. However, for the CNN, their methodology yields better results on all measured metrics. Therefore, the extraction of the MSB appears to result in a better transfer rate of adversarial examples compared to our method. This intuition is confirmed by the results provided in Table 7.7.

Indeed, except for the VGG11 architecture, the methodology proposed by Rakin *et al.*

Table 7.6 – Comparison with [Hec+23].

Method	MLP			CNN		
	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)
[Hec+23]	92.93	96.44	**0.0**	**75.27**	**85.58**	**1.83**
Side-channel-based oracle (Complete)	**97.37**	**98.34**	48.38	65.39	74.86	25.02
Side-channel-based oracle (Single)	-	-	-	61.98	69.56	29.26

Table 7.7 – Comparison with [Rak+21].

Method	ResNet18			ResNet34			VGG11		
	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)	Accuracy (%)	Fidelity (%)	AuA (%)
[Rak+21]	**89.15**	90.64	**1.89**	**88.02**	89.22	**1.69**	**84.67**	86.55	17.2
Side-channel-based oracle	83.57	**93.61**	8.87	83.68	**92.49**	9.95	81.34	**90.16**	10.92
Single side-channel-based oracle	75.69	79.27	26.70	77.93	81.35	26.47	77.41	81.87	21.18

[Rak+21], which is also based on the extraction of the MSB, leads to models with a higher transfer rate compared to ours, despite the fact that our methodology outperforms theirs in terms of fidelity on all architectures. This highlights the absence of a direct link between label agreement and the transfer rate of adversarial examples. Furthermore, we observe that despite having a more realistic threat model, our attack achieves similar results and even outperforms SOTA attacks on complex architectures in terms of fidelity. This shows the potential of side-channel attacks for fidelity-based model extraction of DNNs at the neuron level.

7.4.2 Inconclusive Alternatives to Inequalities

To adapt cryptanalytical extraction to quantization, we test several methods. In this section, we provide a brief description of these methods and our observations on the reasons that may have led to their failure.

Support Vector Machine. We begin by first mentioning an alternative method that is subsequent to our adaptation to inequalities. Indeed, with this method, finding the hyperplane can be solved using supervised learning, as previously mentioned, using the inequalities from the oracle function. In our work, we train a substitute model layer by layer, using gradient descent. An alternative method is to use the Support Vector

Machine (SVM) [CV95] algorithm with linear kernels to learn the hyperplane. Indeed, the intuition is that a subset of the critical points would be sufficiently close to the hyperplanes to construct support vectors and extract the hyperplane. However, we implement this method, and it yields similar results to the approach we presented in this chapter.

Closest Vector Problem. As mentioned in Subsection 7.1.2, the primary issue in cryptanalytical extraction of quantized models is the error in the estimated critical points. This problem is represented on a toy example with two dimensions in Figure 7.4 and can be described accordingly. An attacker tries to estimate the equation of a hyperplane defined on $\mathbb{R}^2$ by searching for critical points in a subset forming a grid on $\mathbb{R}^2$. More formally, this subset corresponds to a lattice in $\mathbb{R}^2$ with its basis defined by the quantization step of the weights. Indeed, for quantized inputs, there exists a basis of vectors such that all inputs can be expressed as a linear combination, with integer coefficients, of this basis. In this case, it is possible to see cryptanalytical extraction as a first step to find a vector and then perform a search for the closest vector in the lattice defined by the weights, which is a well-defined problem.

Integer Linear Programming. Finally, this problem is closely related to integer linear programming, and as such, we test different methods based on this principle. However, the same restriction as for the lattice approach, *i.e*, large size of the vector and knowledge of the quantization step for the weights, applies in this context. Furthermore, the large bias caused by input quantization led to an incorrect solution. In practice, we achieve results similar to those obtained with SVMs and the proposed main methodology.

7.5 Conclusion

In this chapter, we present a new attack to perform fidelity-based model extraction of quantized DNNs. We first demonstrate the link between input quantization and the poor performance of cryptanalytical extraction on quantized DNNs. Then, we propose an alternative methodology, solely based on a physical attack, to target SOTA quantized architectures. We improve the SOTA in terms of fidelity on all targeted architectures in a more realistic threat model. Finally, we argue that a subdivision of the objectives within fidelity-based model extraction attacks is needed. We argue that our method, as well as fault-based methods [Rak+21; Hec+23], operate on a different level compared to crypt-

analytical and traditional learning-based methods, and as such, do not share the same objectives or evaluation metrics.

Despite proving effective on quantized models, our methodology does not replace cryptanalytical extraction, and more generally, extraction at the weight level of quantized DNNs. Therefore, we propose in the final chapter of this thesis to introduce a new methodology based on a profiled attack to perform extraction at the weight level of a quantized DNN.

Chapter 8

Extraction of Quantized Models without a Profiling Device

In the previous chapter, we proposed a fidelity-based model extraction attack at the neuron level using a non-profiled side-channel attack. As previously mentioned, the goal of such attacks is to maximize the fidelity and the transfer of adversarial examples. However, it does not guarantee equivalence at the weights level as cryptanalytic extraction does. This level of extraction is necessary to perform strong white-box attacks, such as training data reconstruction [Hai+22]. Therefore, we argue that it is necessary to propose a methodology for extracting information at the weight level of quantized models. Classical fidelity-based model extraction attacks already require numerous assumptions for the attacker. Therefore, it would be best not to add any further assumptions for the attacker. In this chapter, we introduce a new methodology based on a profiled side-channel attack whose specificity is that it does not require access to an open copy of the targeted device. Indeed, we leverage the observations made in the previous chapter on cryptanalytical extraction to remove the need for a profiling device. Finally, we propose countermeasures to mitigate the attack introduced in this chapter, as well as those introduced in Chapter 6 and in Chapter 7.

Contents

8.1 Overview of the Attack 165

8.1.1 Motivations 165

8.1.2 Extraction of the First Layer 167

8.1.3 Extending the Attack 170

8.1.4 Complete Methodology 172

8.2 Practical Experiments 175

8.2.1 Threat Model and Metrics 175

8.2.2 Experimental Setup and Targeted Architectures 176

8.2.3 Results of the Side-Channel Attack 177
8.2.4 Results of the Complete Attack 181
8.3 Countermeasures 188
8.3.1 Look-Up Table 188
8.3.2 Shuffling the Neurons 189
8.4 Conclusion 191

8.1 Overview of the Attack

8.1.1 Motivations

The purpose of this chapter is to propose a methodology to perform fidelity-based model extraction at the weight level for quantized models. For this reason, the methods based on complete extraction via cryptanalysis are excluded (see Section 7.1). Furthermore, the idea is to provide a generic methodology without increasing the level of attacker knowledge compared to the threat model considered in Chapter 6. Therefore, our methodology should not require previous reverse engineering steps, which makes attacks based on CEMA [Hor+25; Bat+19] impractical, as mentioned in Subsection 4.3.3, and it should not necessitate the use of a profiling device.

Under these constraints, we employ an equations-based methodology, similar to cryptanalytical extraction. As demonstrated in Section 7.1, the issue with cryptanalytical extraction lies in estimating the critical points. Indeed, as quantization becomes more aggressive, the average distance of the critical points to the hyperplanes increases, and the extraction of the weights becomes imprecise. This phenomenon is due to our search for points at the intersection of the neuron-induced hyperplane (see Definition 4.8) and the input space accessible after quantization (see Figure 7.4).

Therefore, a direct solution would be to use random points, *i.e.*, points not on the neuron-induced hyperplane but randomly localized in the input space, to construct the system of equations. Indeed, since these points do not need to be localized on the hyperplane, an attacker, under the assumption that she knows the input quantization, can select points in the quantized input space to construct unbiased equations. However, the issue is that for random points, the solutions to the equations are not determined. Indeed, we recall that by definition (see Definition 4.7), for a critical point $\boldsymbol{X} \in \mathcal{H}_\eta$ associated

with a neuron η with a weight vector $\boldsymbol{\theta}_\eta$ and a bias β_η, we have the following equation:

$$\sum_{i=0}^{n} \boldsymbol{\theta}_{\eta,i} \boldsymbol{X}_i + \beta_\eta = 0, \boldsymbol{X} \in \mathbb{R}^n \tag{8.1}$$

However, if $\boldsymbol{X}$ no longer corresponds to a critical point but to a random point, Equation 8.1 now becomes:

$$\sum_{i=0}^{n} \boldsymbol{\theta}_{\eta,i} \boldsymbol{X}_i + \beta_\eta = A, \boldsymbol{X} \in \mathbb{R}^n \tag{8.2}$$

with A, an unknown real value, if we suppose infinite precision. Therefore, for a random point, the activation value A cannot be determined without knowing the parameters of the neuron, and consequently the system of equations cannot be constructed.

One solution to mitigate this problem is to use a side-channel attack to extract the activation value of a given point, and then, using this attack, construct a system of equations from random points. Such methodology would lead to several advantages:

- **Avoiding the quantization error.** By definition, this methodology does not require critical points, and therefore, under the hypothesis that the attacker knows the input quantization, the methodology is not impacted by errors related to input quantization.
- **Reducing the difficulties with the special-case neurons.** Furthermore, as mentioned in Subsection 6.4.2, in cryptanalytical extraction, certain neurons can be difficult to extract. This is due to the fact that their hyperplanes are localized in regions where the nesting of previous ReLU functions nullifies the input space. However, since this method does not require critical points, an attacker can sample random points across all parts of the input space, not just in specific regions associated with the hyperplanes, which improves both the extraction for special case neurons and also in general, as illustrated in Figure 8.1. In this figure, a neuron in the second layer can only be extracted once critical points in the area where the neuron in the first layer is active, *i.e.*, the blue area in Figure 8.1. Otherwise, the critical point is associated with a deactivated neuron in the preceding layer, which implies that one of its entries is null, and the corresponding weight cannot be extracted. Such points are referred to as incomplete critical points, and critical points localized in regions without deactivated neurons in the preceding layer are referred to as complete critical points. Furthermore, sampling for points on the hyperplane in active regions can become increasingly difficult as the depth of the

model increases. However, by using random points, the attacker no longer has to sample points both on the hyperplane and in the region where the neuron in the previous layer is active. Instead, the attacker simply needs to sample in this active region, which is much easier.

- **Possibility to remove the propagation error.** Finally, the last advantage is related to the iterative nature of equations-based attacks. Since we are constructing a system of equations at each layer, the solution to the preceding layer becomes the input of the next layer, which implies that an error in the solution will be accumulated and propagated throughout the network. We refer to this phenomenon as error propagation. We illustrated its effect on a truncated MobileNetv1 in Subsection 6.4.1. The origin of this error also lies in the search for the critical points. Since we use binary search to find points on the neuron-induced hyperplane, we cannot exactly place a point on the hyperplane, but only within a given distance of it, introducing a small error in the estimation of the weights. However, in the case where we use random points, the initial error is absent. Indeed, as previously mentioned, since we do not use equations related to the hyperplane, we do not need to localize the point precisely on the hyperplane, which removes the estimation error on the critical points. Furthermore, as mentioned in the first point, since we can choose random points anywhere in the input space, we can use points already in the quantized input space, which remove the quantization error on the input. Therefore, in these conditions, no error is expected in the estimated weights. Finally, with access to the activation values, it is also possible to construct the equations not from the estimated solution, but directly using the side-channel attack. Under the assumption that the side-channel attack is perfect, then no error is propagated since each layer is estimated completely independently of the others.

The purpose of this chapter is therefore to construct a side-channel attack that enables the extraction of the activation values without increasing the level of knowledge required from the attacker. To achieve this, we propose to describe in the next section a first attack step that allows an attacker to transition from a black-box to a white-box scenario for the first layer.

8.1.2 Extraction of the First Layer

While in the previous chapter we argued that the difficulties associated with quantization and error propagation made the full extraction of DNNs impractical due to the accu-

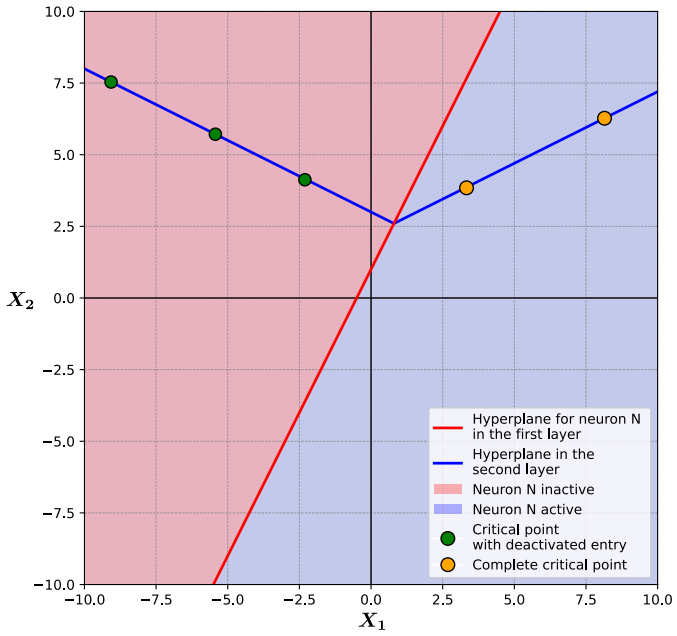


Figure 8.1 – Impact of moving from critical equations to random equations to estimate the weights of a neuron.

mulation of errors from one layer to another, the extraction of one-layer neural networks remains possible. Indeed, in such a scenario, the errors are, by definition, not accumulated from one layer to another, as there is only one layer. Therefore, without increasing the number of queries made to the targeted neural network, compared to classical methodology, it is possible to extract its first layer. As described in Section 7.1, due to quantization errors, some neurons may not be properly extracted, depending on the size of the input vector (see Subsection 7.1.2). However, under the hypothesis that an attacker can determine which neurons have been correctly extracted, the attacker gains access to a set of neurons and their respective parameters in white-box settings after cryptanalytical extraction. To validate this hypothesis, we conduct a simulated extraction of the first layer of a quantized DNN, similarly to what has been done in Subsection 7.1.1.

We target the first 50 neurons of the first layer of 9 MLPs with the same architecture, but with varying input sizes. In practice, the architecture follows this pattern: I-64-32-16-10, with I the input dimension. We use the same notation as in Chapter 6, and as such, an MLP 10-64-32-16-10 corresponds to a model with an input size of 10 with 3 hidden layers composed of 64, 32, and 16 neurons and an output size of 10. The MLPs are trained on a random task using the TensorFlow framework and fully quantized to 8 bits by the

NNoM framework [Ma20]. We compile the source code associated with each model and integrate the compiled library into a Python script that performs the extraction of the first layer neuron by neuron. However, since we are only interested in whether or not we move from a black-box scenario to a white-box scenario, we measure the number of neurons for which the extraction yields the correct signature (see Definition 4.9). To perform this extraction, we utilize the same algorithm as the one introduced in Algorithm 2 to search for the critical points. The use of the NNoM framework with Algorithm 2 is coherent, as we have detailed in Subsection 7.3.2 that an oracle function similar to the one used in Subsection 6.2.4 can be constructed (see Subsection 7.3.2). We modify the convergence criterion of Algorithm 2 to take into account the input quantization. In this experiment, the algorithm stops when the generated new point is equal, after quantization, to one of the previous points, either $\boldsymbol{M} = \boldsymbol{X}$ or $\boldsymbol{M} = \boldsymbol{Y}$ in Algorithm 2. This criterion is more robust to input quantization compared to a threshold on the distance between $\boldsymbol{X}$ and $\boldsymbol{Y}$. It ensures that for the first layer of an MLP, the number of requests per parameter will remain, on average, the same across our experiment, as we can observe in Figure 8.2. We average the results by repeating the extraction process five times and present the results in Figure 8.2.

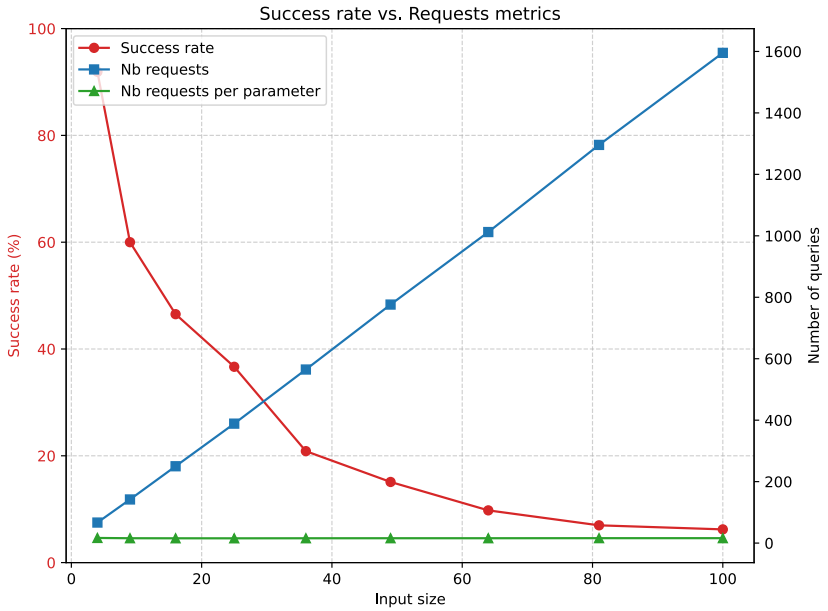


Figure 8.2 – Results of our simulated extraction of the first layer of quantized MLP using the NNoM framework.

As expected, we observe in Figure 8.2 that the percentage of perfectly extracted neurons diminishes greatly as the input size increases, which is consistent with the results from Subsection 7.1.2. However, even for an input size of 10 by 10, which is larger than the input size of several SOTA architectures such as VGG11, MobileNetv1, or ResNet18, which all use $3 \times 3 \times 3$ filters in the first layer, the number of perfectly extracted neurons is not null. Therefore, it is possible to use cryptanalytical extraction to move from a black-box scenario to a white-box scenario for the first layer.

In the following section, we show how we can use this result to our advantage to perform a complete extraction of a quantized DNN.

8.1.3 Extending the Attack

The cryptanalytical extraction of the first layer is based on an oracle function that indicates whether two neurons are in the same state for two given inputs (see Section 6.2). In the rest of this chapter, we present our results under the assumption that the oracle function returns perfectly the state of all neurons. As shown in the previous section, this oracle function can then be used to extract the weights of the first layer, enabling an attacker to operate in white-box settings for this layer. However, this result is only useful for an attacker under three conditions:

- **Neuron Selection.** We have shown in Figure 8.2 that even for small MLPs, not all neurons are perfectly extracted. Therefore, an attacker is truly in a white-box setting if and only if they can detect which neurons are correctly extracted.
- **Extending the attack to other layers.** Furthermore, as mentioned in Chapter 7, cryptanalytical extraction cannot be applied to the succeeding layers. However, the attacker can use the white-box settings to introduce a new profiled side-channel attack on deeper layers.
- **Impact of the scaling factor.** Finally, cryptanalytical extraction only extracts a signature and not the true weights of the neuron. Therefore, an attacker must extract the value of the scaling factor to be truly in a white-box setting.

In this section, we describe how we address these conditions, starting with the identification of the correctly extracted neuron and then detailing the profiled side-channel attack under the assumption that the scaling factor has been successfully extracted. We conclude by introducing a methodology for extracting the scaling factor.

Neuron selection. To detect the neurons that have been correctly extracted, we simply test if the extracted hyperplane is coherent with the state of the critical points obtained during the cryptanalytical extraction. Since their state is already known, as they were already obtained during the binary search leading to their extraction (see Subsection 6.2.1), this method does not add any queries to the overall attack. The attacker simply computes sequentially the dot product of each critical point with the estimated signature and compares it with the state obtained. We test this methodology on the same dataset as in Subsection 8.1.2 and achieve a 100% success rate in distinguishing correctly extracted neurons from incorrect ones.

Extending the attack to other layers. Under the assumption that the scaling factor is known, the attacker is in white-box settings and knows both the parameter values and the activation value for any input. To avoid the issues mentioned in [Jou+23a], *e.g.*, the initial value of the accumulated sum, the order of the multiplication between the weight and the input, we use a profiled side-channel attack on the ReLU function, this time, not targeting the sign, but the processed activation value. Since the parameters of the matrix product are known, it is possible to determine an input set that results in a uniform distribution of the activation values, thereby improving the profiling phase performance. Furthermore, under the assumption that the processing is the same for all neurons in all layers, an attacker can only profile one neuron in the first layer and use the constructed model on all neurons. As shown in Figure 8.2, even without increasing the number of queries for cryptanalytical extraction, at least one neuron is successfully extracted in our experiment, and thus, it is possible to perform the profiling phase using this neuron under the assumption that such a neuron can be identified (see Neuron selection phase).

Impact of the scaling factor. The last issue is the extraction of the scaling factor. We propose two complementary solutions to deal with this issue. The first one is based on the fact that we target an open-source framework, and as such, we know the type of quantization used, *i.e.*, the weights are quantized to 8-bit signed integers. This enables an attacker to restrict the number of hypotheses on the scaling factor, as the true weights must satisfy this condition. However, this does not definitely provide the value, and as such, we use an exhaustive search on the scaling factor to recover its true value. We conduct independent profiling phases based on the same set of traces, but based on all possible values of the scaling factor, and apply them to all remaining correctly extracted

neurons from the first layer. We then select the scaling factor leading to the best solution for all remaining neurons.

Using profiled side-channel attacks to target the activation values allows an attacker to target quantized DNNs while considering a similar threat model as the one presented in Chapter 6. In the following section, we describe the complete workflow of our attack.

8.1.4 Complete Methodology

In this section, we outline our methodology for extracting DNNs quantized using the NNoM framework. Our attack is based on five steps:

- **Stage 1: Cryptanalytic extraction of the first layer.** In the first step, we use the same methodology introduced in Section 6.2 to extract the critical points and the signatures of all neurons in the first layer. This step is illustrated in Figure 8.3.

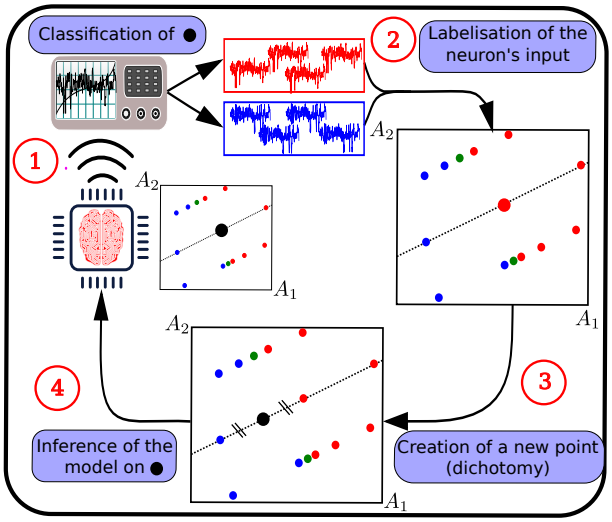


Figure 8.3 – First step of our methodology performing cryptanalytical extraction of the first layer.

- **Stage 2: Filtering the incorrectly extracted neurons.** We then use the critical points to test the validity of the estimated signatures based on their state as described in Subsection 8.1.3. We provide a schematic description of this step in Figure 8.4.
- **Stage 3: Profiling one neuron.** In the third step, illustrated in Figure 8.5, we perform the profiling phase on one of the correctly extracted neurons after Step 2.

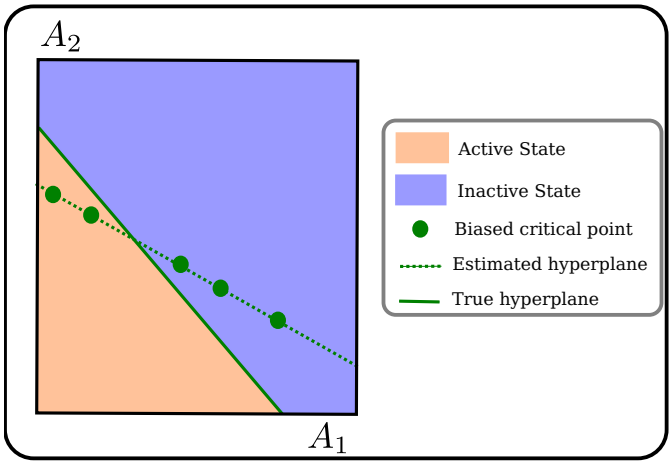


Figure 8.4 – Filtering our the incorrect signature based on the state of the critical point in the second stage of our framework.

Since we target one of the correctly extracted neurons, we know its signature, and as such, we can compute the activation values of this neuron up to a scaling factor as described in Subsection 8.1.3. Therefore, we perform multiple profiling phases on the activation values independently, for all possible values of the scaling factor. To conduct this step, we construct a dataset of inputs such that the distribution of activation values is uniform. We then send this input to the targeted device and collect the associated side-channel traces. Once all traces have been acquired, we construct the models based on the known activation values depending on the values of the scaling factor.

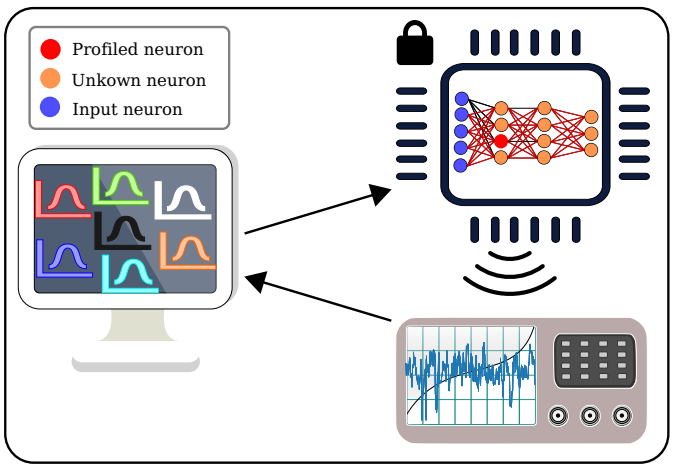


Figure 8.5 – Profiling one neuron on the targeted device to extract the activation values for different scaling factor values.

- **Stage 4: Selection of the correct scaling factor.** In the fourth step, we use the estimated models to target the neurons that have already been successfully extracted in the first layer. Using these models, we retrieve the activation values for a set of known inputs, and we construct unbiased systems of equations directly connected to the activation values. Since each model is associated with a hypothesis on the scaling factor, the activation values obtained from the profiled attack are also associated with the same hypothesis, and therefore each system is associated with a given hypothesis on the scaling factor (see Subsection 8.1.3). We then select the hypothesis associated with the best solutions compared to the one obtained through cryptanalytical extraction, as shown in Figure 8.6.

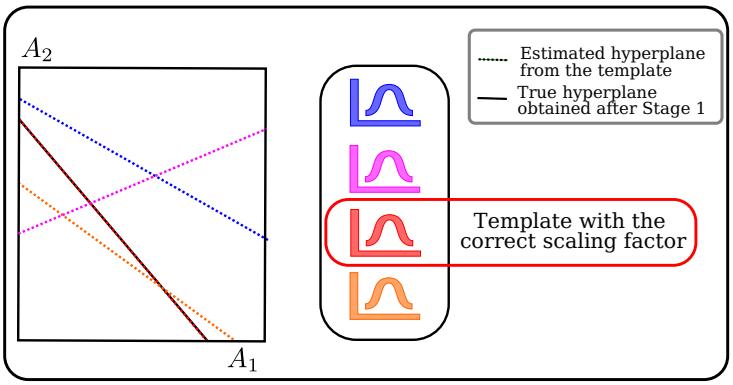


Figure 8.6 – Schematic representation of the extraction of the scaling factor by selecting the model leading to the best results.

- **Stage 5: Sequential extraction of all the remaining neurons.** Finally, now that we have the correctly estimated model, we iteratively extract all neurons in the following layers. For this step, the attacker sends random inputs to the targeted device, then uses the model to extract the activation values. The attacker constructs a system of equations based on these values for each neuron layer by layer, similarly to what has been described in Subsection 6.2.2. The final step of our attack is represented in Figure 8.7.

One key takeaway from this methodology is that, despite conducting a profiled side-channel attack, which typically requires a profiling device (see Definition 3.7), this is not the case in our scenario. Indeed, due to attack steps 1 and 2 of our methodology, we can extract the signature of at least one neuron and use this signature to infer the activation

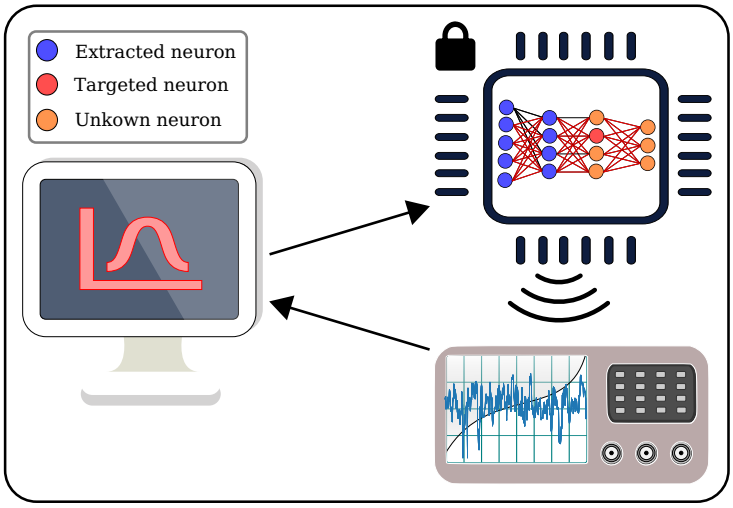


Figure 8.7 – Illustration of Stage 5 corresponding to the iterative extraction of the remaining neurons.

values for any input vector, under the assumption that the input quantization is known. Therefore, we can use the targeted device directly as a profiling device, completely circumventing the necessity for an attacker to have access to an open device. In consequence, the assumptions for step 3 are directly fulfilled even in the same threat model as in Chapter 6. However, the first two steps of our framework require that the attacker knows the type of quantization used by the targeted DNN. We provide a detailed description of our threat model in the following section and then present our experimental results.

8.2 Practical Experiments

8.2.1 Threat Model and Metrics

In this chapter, we consider an attacker whose goal is to perform a fidelity-based model extraction of a quantized model. However, following the categories introduced in Subsection 7.2.4, we consider more specifically an attacker aiming for extraction at the weight level. In this sense, our attack is closer to the objective of the attacker considered in Chapter 6. As such, we consider two metrics in evaluating our attack: the maximum error between the estimated and true weights and the number of queries made to the targeted model.

In terms of the threat model, we follow the same approach as in Chapter 6, *i.e.*, the attacker can send an unlimited number of queries to the targeted model, and in return,

the model outputs a hard-label. During the inference of the targeted model, the attacker is supposed to have physical access to the targeted device. Similarly to Chapter 6, we also assume that the attacker only targets Deep-ReLU networks. However, in this chapter, the targeted models are fully quantized to 8 bits. Furthermore, the quantization of the processed values, and more specifically, of the inputs and the weights, is supposed to be known by the attacker. As presented in Subsection 8.1.4, we do not require a profiling device. Therefore, our attack corresponds to a non-profiled gray box attack.

8.2.2 Experimental Setup and Targeted Architectures

Our experiment directly follows the one presented in Chapter 7, and as such, we use the same experimental settings except for the triggers used in the side-channel attack. More specifically, we target quantized DNNs by the NNOM framework [Ma20], embedded on an ARM Cortex-M7 MCU on an STM32-F767 board. Similarly to what is presented in Subsection 7.3.1, we also target the modified constant-time implementation of the ReLU function and compile the project without compiler optimization.

Finally, we use a similar setup, consisting of a probe from Langer (EMV-Technik RF-U 2.5 with a frequency range of 30 MHz to 3 GHz) as well as an amplifier (2X60-3018G with a 12V supply), and a low-pass filter with a cutoff frequency of 270 MHz. To capture the EM side-channel leakages, we use a Lecroy oscilloscope (2.5 GHz WaveRunner 625Zi). However, to improve the alignment of the side-channel traces, we add a more precise trigger compared to Subsection 7.3.1. Previously, the configuration of the trigger was not considered difficult, as it was only set at the beginning of the ReLU function, whose patterns differ from the other processing patterns. In this chapter, we modify the implementation to trigger within the execution of the ReLU function on each targeted neuron. Furthermore, we also modify the implementation of the ReLU function to force the target to execute NOP instructions before and after the processing of the activation values.

To validate our methodology, we first test it on simple architectures. As such, we base our choice of target on the SOTA and extract the MLPs used to demonstrate the validity of cryptanalytical extraction in [CJM20]. We target four MLPs with the following architectures: 10-10-10-1, 10-20-20-1, 40-20-10-10-1, and 80-40-20-1.

Finally, to avoid overflow of the activation value in the model, NNOM uses a modified implementation of the matrix product. More specifically, NNOM performs a classical matrix multiplication followed by a shift, *i.e.*, a division, of the activation values. The scale

of this shift is set in the configuration file. To avoid underrepresentation of certain activation values during the profiling phase, we set the shift factor to 0 in our experiments. We leave the design of profiled side-channel attacks with unbalanced datasets for future work.

In the rest of this section, we first present the results of the profiled side-channel attack. Then, we evaluate our attack against the presented architectures.

8.2.3 Results of the Side-Channel Attack

As described in Subsection 8.1.4, our attack is based on two different side-channel attacks. The first one is used for cryptanalytical extraction of the first layer and consists of extracting the sign of the neurons. And the second one corresponds to the profiled side-channel attacks targeting the value of the activation function during the processing of the ReLU function. As we have already demonstrated the viability of the first attack in a more difficult threat model, *i.e.*, less precise trigger, we will not further describe this part of the attack and only present, in this section, the result from the profiled side-channel attack. As such, we directly consider in this section that we have extracted the first layer of the targeted DNN with the same success rate as in Subsection 8.1.2, *i.e.*, we suppose that Steps 1 and 2 have already been performed (see Subsection 8.1.2), and we now validate our methodology starting from Step 3.

Construction of the sets. To conduct our profiled side-channel attack, we follow the methodology introduced in Subsection 3.2.3. We first collect two sets of traces corresponding to the profiling set and the attack set. For the profiling set, we collect 10^6 EM traces corresponding to the processing of the ReLU function. To avoid the issue of unbalanced classes in the profiling dataset mentioned in Subsection 5.2.2, we use a set of inputs such that the distribution of the activation value is uniform in the training set. Since we are profiling a neuron in the first layer for which the parameters are known, the search for such a set is straightforward. To measure the performance of our profiled attack with respect to the number of attack traces, we collect $1,000$ EM traces for each activation value in the attack set. Furthermore, to ensure that no bias is associated with one specific hypothesis of the activation value, we validate our attack on all hypotheses. Since we target 8-bit variables, this corresponds to testing the attack on all 256 possible classes, and therefore, we acquire $256,000$ EM traces in total for the attack set. By doing so, we construct our attack dataset on fixed data, which should improve the performance of the

profiled side-channel attack.

Preprocessing. Before conducting the side-channel attack, we preprocess both sets. We first identify the temporal localization corresponding to the manipulation of the activation values in the signal by computing the SNR with respect to the activation values. We include the SNR without aligning the traces on the profiling set in Figure 8.8.

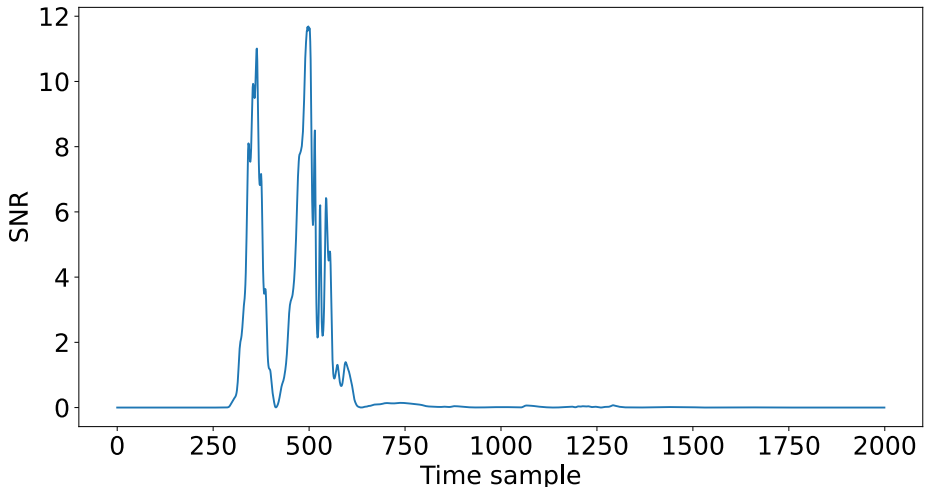
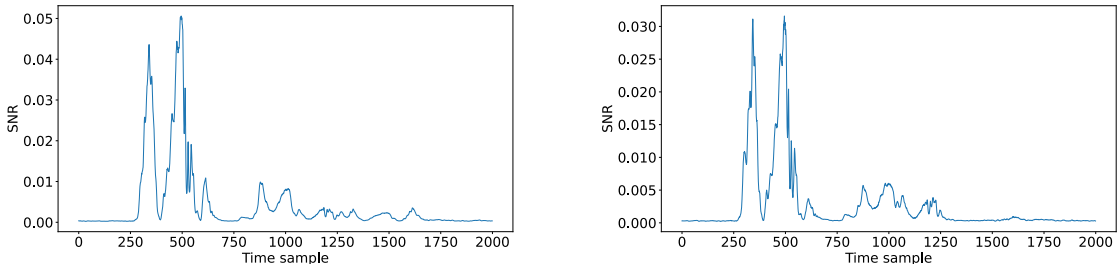


Figure 8.8 – SNR on the activation value on the complete profiling set.

Since we know from the experiments in Chapter 6 and Chapter 7 that the sign of the activation leaks particularly, we decide to subdivide both the profiling and attack set into two subsets corresponding to the case where the neuron is active or inactive. We then recompute the SNR for both groups on the profiling set in Figure 8.9a and Figure 8.9b.

As expected, the maximum value drops significantly compared to the results computed on both groups, showing the impact of the bit sign. We then remove approximately 20% of the traces by excluding those displaying a strong average difference during the processing of the activation values. Indeed, we observe in the side-channel traces the occurrence of a perturbation that we are unable to filter or identify the cause of in the source code. Therefore, after removing those traces and splitting into two groups, we obtain two final profiling datasets composed of $413,717$ traces (resp. $411,765$) for the first (resp. second) group, and of $93,486$ (resp. $92,775$) traces for group 1 (resp. 2) in the attack set. We perform the attack in both the temporal and the frequency domains via a spectrogram using



(a) SNR on the positive activation values in the sub-group of the profiling set.

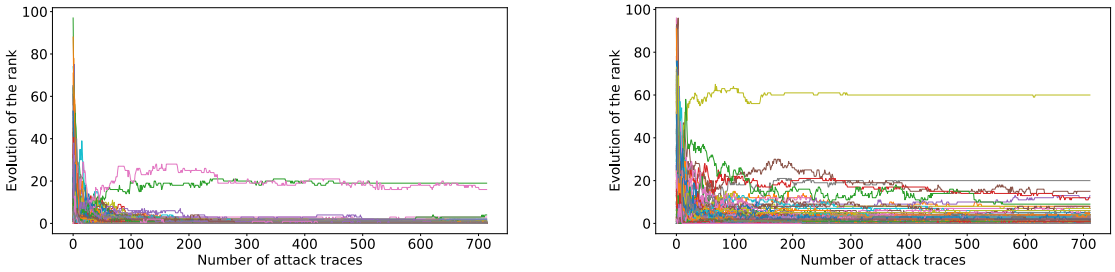
(b) SNR on the negative activation values in the sub-group of the profiling set.

Figure 8.9 – Evolution of the SNR on the activity of the neuron for the active and inactive subgroups.

a Fourier transform with a window set to 300 samples in our experiment and a sampling frequency set to 3 along the time axis. In both domains, we reduce the dimension of each trace by selecting a subset of 500 PoI (see Definition 3.5).

Results of the attack on the profiled neuron. In our experiment, we use a template-based approach (see Subsection 3.2.3) for the profiled attack. We construct the templates on the fifth neuron of the first layer. The choice of this neuron is arbitrary. The only condition is that the profiled neuron must be correctly extracted (see Subsection 8.1.4). In this case, it corresponds to one of the correctly extracted neurons after the first step of our methodology (see Figure 8.3). The templates are constructed, and the attack is performed on each group independently, following the same methodology and evaluated using the same metrics. For both groups, for each possible activation value, we measure the evolution of the rank (see Definition 3.10) of the correct hypothesis with respect to the number of attack traces. We present this evolution for group 1 (resp. 2) in Figure 8.10a (resp. Figure 8.10b).

If the final rank equals 0, then the attacker has extracted the correct value. We measure the success rate (see Definition 3.11) of our attack as the percentage of correctly extracted values. We only include the results in the frequency domain as these correspond to the highest success rate. For group 1, we obtain a success rate of 81.25%, and for group 2, we achieve a success rate of 39.84%. While the success rate for group 2 is significantly lower than that of group 1, we argue that the attack remains practical in this context. Indeed, since we can retrieve the activation value with a high success rate for group 1, we can



(a) Rank evolution for group 1.

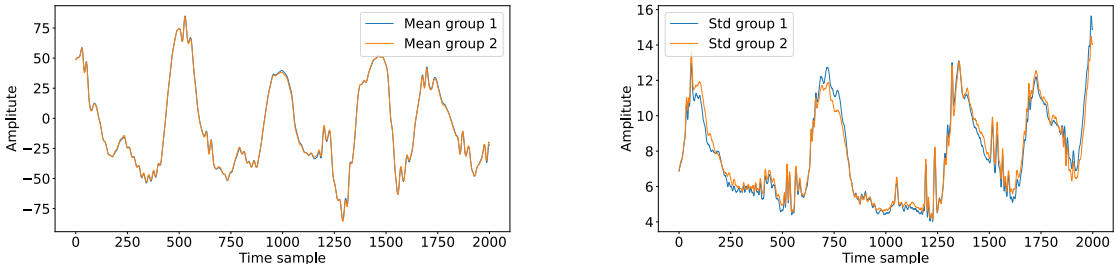
(b) Rank evolution for group 2.

Figure 8.10 – Evolution of the rank for each possible hypothesis in function of the number of attack traces.

directly construct the system of equations based on inputs solely corresponding to this group. We provide supplementary details in Subsection 8.2.4.

Attacking different neurons. The results presented in the preceding paragraph are obtained on the same neuron as the one on which the profiling was done. We decide to validate that the processing remains the same from one neuron to another by attacking a different neuron. In practice, we target neuron 55 using the templates constructed for neuron 5. We use the same experimental setup and acquire the same number of attack traces as for neuron 5, *i.e.*, 256,000, before applying the preprocessing. We compute the same metric as for the previous experiment, *i.e.*, the success rate. However, in this context, we achieve a success rate of only 2.34% for both groups, improving random guess by only 1.56%. To identify the cause of this change, we compare the statistical characteristics of the traces of the attack set of neurons 5 and 55. More specifically, we compute the mean and the standard deviation of both datasets at each time sample. We include the results of this analysis in Figure 8.11.

In both figures, especially for the standard deviation in Figure 8.11b, we observe a difference between the two averaged traces. This difference implies that there is a change in the processing of the ReLU function between the two neurons. However, after analyzing the source code and assembly code, we were unable to identify the cause of this change. We decide to further preprocess our traces by standardizing the traceset. After standardization, the best results are obtained in the temporal domain, and we achieve a success rate of 7.81% (resp. 10.16%) for group 1 (resp. 2). While this success rate is very low, we decided in this thesis, to present the simulated results of the complete extraction obtained under the assumption that the processing remains the same from one neuron to



(a) Temporal evolution of the mean for both neurons.

(b) Temporal evolution of the standard deviation for both neurons.

Figure 8.11 – Statistical analysis of the attack set corresponding to neurons 5 and 55.

another, and leave the improvement of the side-channel attack for future work.

8.2.4 Results of the Complete Attack

Despite the results from Subsection 8.2.3, we conduct our methodology under the assumption that the processing is the same for all neurons. As such, we simulate the side-channel attack by offering direct access to the activation value of the targeted DNN. However, to take into account the results from Subsection 8.2.3, we use a biased estimation of the activation value. More specifically, in our simulation of the complete attack, we consider that an attacker can only determine the activation value with a probability of 0.8125 if the neuron is active. This probability directly originates from the results from Subsection 8.2.3 for group 1, *i.e.*, active neurons. When the estimation fails, the activation value is chosen randomly among the rest of the possible values for the group, *i.e.*, 127 possible values. To remain consistent with the experiment in Subsection 8.2.3, we consider that the estimation of the activation value requires 10^3 queries to the targeted model.

The purpose of this section is to demonstrate the viability of our methodology, justifying the need for future work improving the profiled side-channel attack on the activation value. However, due to the fact that the processing is different between the two neurons, we are unable to validate the methodology for the extraction of the scaling factor (see Subsection 8.1.3). Similarly to the side-channel attack, we suppose that the scaling factor has been previously extracted in the rest of this section. In the rest of this section, all simulations are conducted similarly to those presented in Subsection 8.1.2, *i.e.*, the simulations are performed on DNNs fully quantized to 8 bits using the NNOM framework and

compiled into a library.

Handling the side-channel error. As presented in Subsection 8.1.4, we construct a system of equations based on the estimated activation values. However, due to the performance of the side-channel attack, the estimation is biased. Therefore, the system is also biased. The issue is that, due to quantization, an attacker cannot afford any error in the system of equations; otherwise, the attack fails. A solution is to adapt the RANSAC algorithm [FB81] to our problem. The purpose of this algorithm is to estimate the parameters of a model from a dataset composed of correct data (inliers) and abnormal data (outliers). The algorithm iteratively selects a random subsample of the data and estimates the parameters of the model, *i.e.*, the value of the weights and the bias of the targeted neuron in our case, from this subsample. Then, if the solution satisfies a certain criterion, for example, the residuals of the solution or the number of points considered as inliers based on the estimated solution, the algorithm stops. Otherwise, a new subsample of points is selected. The intuition behind the algorithm is that if enough equations are sampled with a probability p that the equation corresponds to an inlier, then it is possible to find a subset composed of only inliers. The algorithm stops after a given number of iterations. This hyperparameter of the algorithm, combined with the criterion, avoids the exhaustive search over the total number of possible permutations.

We use this algorithm to improve the extraction of the weights. In our case, we directly use the residuals as a first criterion. Indeed, due to the quantization to integer values, an incorrect solution leads to a large, and therefore distinguishable, residual value. To validate this methodology, we simulate the extraction of the first layer of a DNN for different values of p, the success rate of the side-channel attack. As previously mentioned, we are only interested in exact solutions, *i.e.*, perfect extraction of the parameters of the neurons. Therefore, we measure the success rate of our algorithm as the percentage of perfectly extracted neurons as a function of the number of equations sampled. We present the results in Figure 8.12.

We target the first 64 neurons of an MLP, each with 16 weights and no bias. The number of equations starts as the minimal number of equations needed. For each point of the curve, we average the results of 5 different experiments. We observe in Figure 8.12 that using the RANSAC algorithm, it is possible to perfectly extract all neurons even with an imperfect estimation of the activation values. Furthermore, we add a second criterion to improve the RANSAC algorithm. We still keep the value of the residuals as a first

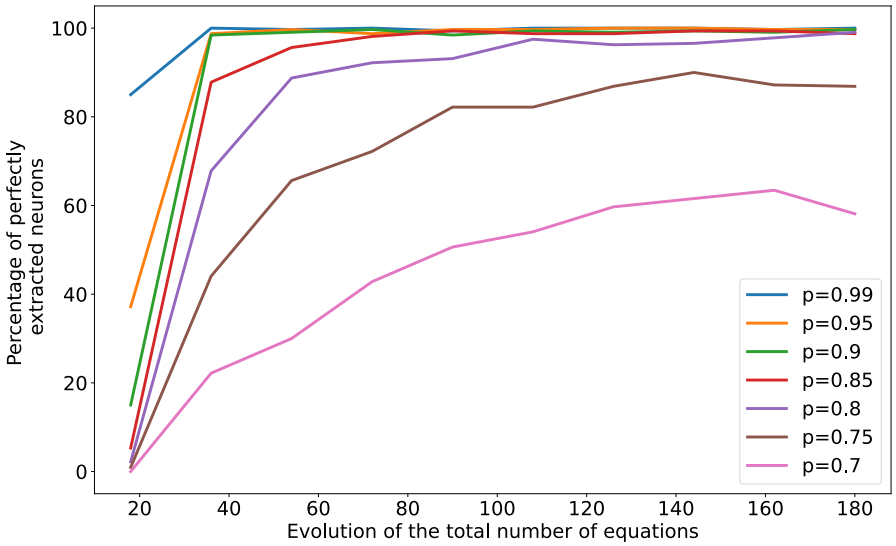


Figure 8.12 – Evolution of the percentage of successfully extracted neurons with the RANSAC algorithms for different initial probabilities.

criterion, but we add a test on the state of critical points. This criterion is inspired by the search for the correct neuron in the extraction of the first layer (see Subsection 8.1.3). For each neuron, we use our traditional binary search algorithm to search for points close to the hyperplane of the targeted neuron. In practice, we sample for 100 points, then we use these points and their state, since they are not exactly on the hyperplane due to the quantization, to test the solution returned by the RANSAC algorithm. In case no solution is found, we can simply sample for more equations and restart the search for the parameters. We validate this solution in the same settings as the previous experiment and obtain a perfect extraction of all neurons. However, this comes at the cost of increased complexity compared to the basic RANSAC algorithm. In Figure 8.13, we illustrate the evolution of the number of queries to maintain a perfect extraction of all neurons using our modified algorithm. The simulation is performed in the same settings as the previous experiment, *i.e.*, same average methodology. We observe in Figure 8.13 an exponential decrease in the number of queries as a function of the success probability of the side-channel attack. This highlights the strong dependency between the side-channel attack and the query complexity of our attack. We further delve into an analysis of the complexity of our attack in the final part of this section.

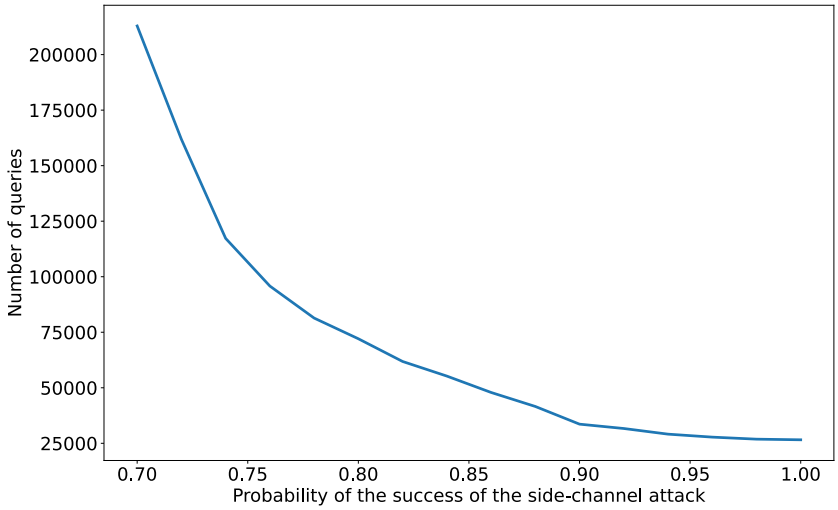


Figure 8.13 – Evolution of the number of queries to achieve a perfect extraction of all 64 neurons using the updated RANSAC algorithm.

Extraction of MLPs. Now that we have validated that we can extract neurons even with our imperfect side-channel attacks, we apply this result to extract complete DNNs. We apply our methodology (see Subsection 8.1.4) to all layers in the targeted DNNs, except for the final one, due to the absence of the ReLU function, as described in Chapter 6. We group the results of our attack on the architecture presented in Subsection 8.2.2 in Table 8.1. As expected, we extract perfectly all hidden layers using our methodology. This

Table 8.1 – Evaluation of the performance of our attack on MLPs quantized to 8 bits using the NNoM framework [Ma20].

Architecture	**Parameters**	**Queries**	$\mathbf{max}\lvert\Delta_\theta\rvert^{L-1}$
MLP 10-10-10-1	**210**	$2^{18.17}$	0
MLP 10-20-20-1	**620**	$2^{23.17}$	0
MLP 40-20-10-10-1	**1,110**	$2^{26.90}$	0
MLP 80-40-20-1	**4,020**	–	–

is due to the fact that each neuron is extracted using an unbiased system, thereby eliminating both the quantization error and the intrinsic error in the estimation of the critical points resulting from the binary search. This validates our methodology and shows that by using random equations instead of critical ones, we can remove the issue associated with quantization and the depth of DNNs.

Compared to traditional cryptanalytical extraction, our method is significantly more expensive in terms of queries. Indeed, on average, our method is $2^{6.7}$ times more expensive in terms of the number of queries. Furthermore, in Table 8.1, we only report the variable part of the number of queries, *i.e.*, we do not include the $1,000,000$ queries associated with the profiling of the best neuron. We choose to exclude this component to improve the comparison of results between the different targeted architectures and previous methods. However, most of the queries in our attack are associated with equations corresponding to potential saturation, *i.e*, equations where the values are either equal to -128 or 127, which are unusable in our context. Indeed, due to the quantization to 8-bit values, there is no way for an attacker to know if these equations are correct or if the activation value has been rounded to one of these values. Therefore, we exclude such points from our system of equations, which increases the complexity of the attack, especially after the first layer. Furthermore, this phenomenon is amplified by the fact that we have set the shift factor to 0 to ensure a correct balance between classes. We expect that once we resolve this issue, the complexity of the total attack should be significantly reduced. We further analyze the theoretical complexity of our attack in the final part of this section.

Moreover, even if our methodology is more expensive in terms of the number of queries compared to cryptanalytical extraction, we argue that it is more efficient due to the potential parallelization of the attack. Indeed, in cryptanalytical extraction, due to the binary search, each query is specifically associated with one neuron. With this method, only a small fraction of the total number of queries is associated with a single neuron, specifically corresponding to the extraction of the first layer and the determination of the criterion points in the RANSAC algorithm. However, the remaining queries can be used independently of the targeted neuron. As such, an attacker can parallelize the extraction of the neurons in the targeted DNN, reducing the window of opportunity required by the attacker on the targeted device.

Finally, we do not include the result on the extraction of the MLP 80-40-20-1, as our attack does not converge to a solution for the neurons in the first layer that are not successfully extracted using cryptanalytical extraction, *i.e.*, Step 1 of our methodology. In the following section, we provide an analysis of the complexity of our attack and present a limitation of our methodology, which explains our difficulties with the MLP 80-40-20-1.

Complexity of the attack. Our attack leverages the RANSAC algorithm to extract the parameters of a neuron from a system of equations that includes both correct and biased equations. As previously mentioned, this is done by iteratively sampling a subset of equations to construct an unbiased system. However, when the probability that a random equation is unbiased becomes lower, or when the size of the system of equations increases, the probability of sampling a correct subset of equations diminishes as well. In particular, the minimal number of subsamples N_{RANSAC} needed to ensure that an unbiased system of equations is found with a probability superior to $1 - \epsilon$ is bounded by:

$$N_{RANSAC} > \frac{\texttt{log}(\epsilon)}{\texttt{log}\left(1 - p_u^S\right)} \tag{8.3}$$

with p_u the probability that a random equation is unbiased and S the minimum number of equations to estimate the parameters of the model. For example, in the configuration presented for extracting the remaining neurons within the first layer after Stage 1 of our framework on the MLP 80-40-20-1, we need to have $N_{RANSAC} > 75,400,825$ to ensure that a solution is found with a probability greater than 0.99. While the iteration of the RANSAC algorithm can be parallelized, this result shows how the attack becomes impractical in such a scenario. To measure the limits of our attack, we decide to evaluate the value of N_{RANSAC} in various system size configurations and success probabilities of the side-channel attack. We regroup the results on a logarithmic scale in Figure 8.14.

In Figure 8.14, we can clearly observe that as the system size increases, the probability must also increase to keep our attack practical. This result highlights the computational limitation of our method when the extraction of activation values is not perfect. A potential solution would be to filter out the equations that are not correctly extracted, either by observing the evolution of the rank of the estimated value, correct or incorrect, with the number of attack traces, or by setting a threshold on the final probability of the side-channel attack. However, this solution is also impractical, as it excludes a large group of values, which can make extracting neurons that only return these values impossible. Therefore, our solution must be optimized to improve the temporal complexity of our attack.

However, in terms of query complexity, if we exclude the queries associated with saturated activation values, we argue that our attack is rather efficient. Indeed, an attacker needs only to sample enough equations n to ensure that there are enough correct equations to estimate a system of size S. Furthermore, if we assume that each side-channel attack

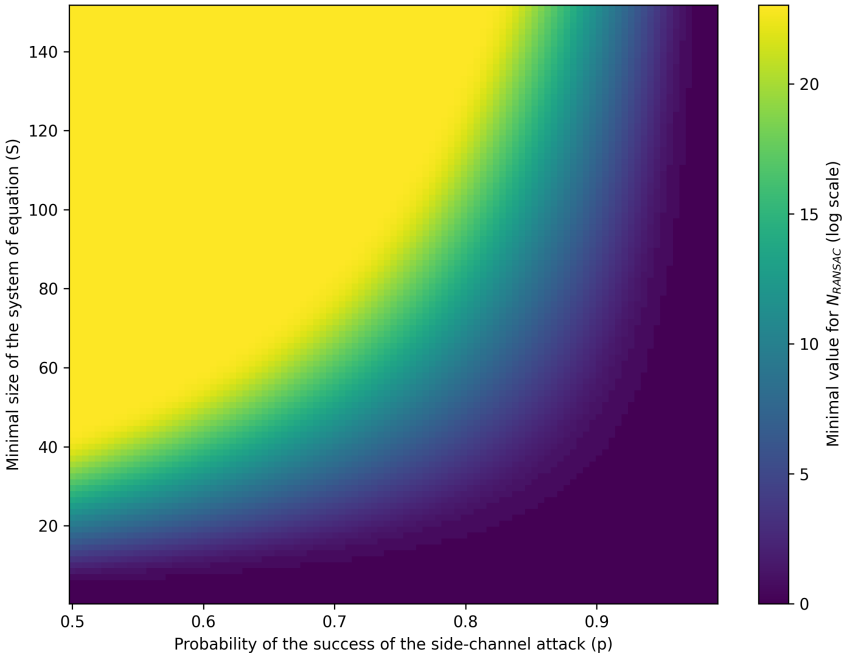


Figure 8.14 – Evolution of the number of iterations of the RANSAC algorithm for different configurations.

on the activation value corresponds to an independent Bernoulli trial with a probability p of success, then the variable X corresponding to the number of unbiased equations within n trials, *i.e.*, the total number of equations, follows a binomial distribution $X \sim \mathcal{B}(n, p)$. And as such, we can estimate a minimum value n of the number of equations to sample to guarantee that enough unbiased equations have been found with a probability superior to ϵ as:

$$\sum_{k=S}^{n} \binom{n}{k} p^k (1-p)^{n-k} > \epsilon \tag{8.4}$$

Therefore, using this bound, it is possible to set a maximum threshold corresponding to the maximum number of equations needed to ensure that the neuron with the highest number of parameters, *i.e.*, the one corresponding to the largest system size, is extracted.

Since we can use the same inputs to create equations for all neurons, in an ideal scenario with a perfect side-channel attack operating on all neurons simultaneously, it is possible to extract all neurons using just this set of inputs. For example, in such a perfect scenario, the extraction of the MLP 80-40-20-1 would only require 111 different inputs

after the first stage of our framework to extract the complete model. Furthermore, once the equations are obtained via the profiled side-channel attack, the attacker can simply perform the RANSAC algorithm to extract all neurons without necessitating access to the physical device.

In this section, we discuss the potential benefits and limitations of our methodology. However, our methodology is based on the assumption that the first layer can be extracted using a first side-channel attack: the extraction of the sign of the targeted neuron. Therefore, in the next section, we propose two side-channel countermeasures to protect DNNs against this attack.

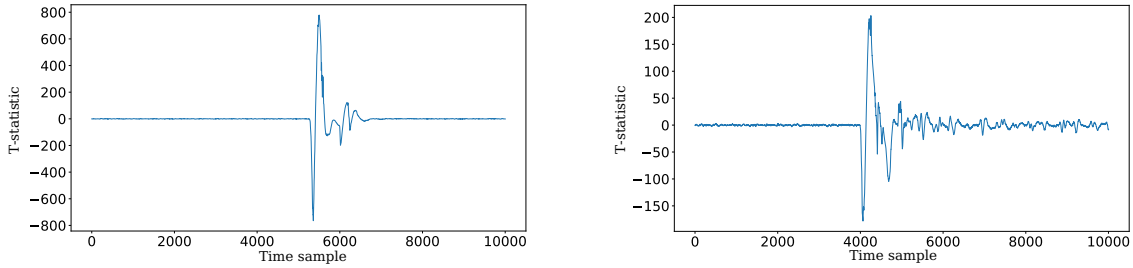
8.3 Countermeasures

Our complete methodology for extracting deeper layers is based on the assumption that we can perform a profiled side-channel attack without a profiling device by extracting the first layer using the methodology introduced in Subsection 6.2.1. Therefore, if this step is not possible, then the methodology presented in this chapter is no longer viable. As such, we aim to protect the processing of the ReLU function against side-channel attacks targeting the sign of the activation value. Furthermore, if we successfully propose a countermeasure for this attack, then the attacks presented in Chapter 6 and Chapter 7 will also become impossible, as they are all based on this vulnerability.

8.3.1 Look-Up Table

In the implementation proposed in [MBC21], the sign bit is used as a mask to set the activation values to 0 or to apply the identity function. While this mask enables a constant-time implementation, the high difference between the Hamming weights of the two masks, corresponding to the two states, introduces a strong correlation between the amplitude of the side-channel traces and the state of the neurons. Therefore, we change the implementation of the ReLU function to introduce a first countermeasure. We implement the ReLU function as a look-up table in our code, where the index of negative values corresponds to a null entry and the rest correspond to an identity mapping. We validate that using this implementation, the behavior of the model does not change by measuring

its accuracy before and after implementing this countermeasure. Finally, we measure the overhead introduced by this function in terms of execution time. This countermeasure increases the average execution time by 0.1 ms, which corresponds to an overhead of 0.1% of the total execution time. We validate this countermeasure by performing a t-test on the sign during the processing of the ReLU function. We report the results with and without the countermeasure in Figure 8.15.



(a) T-test on a constant time implementation of the ReLU function.

(b) T-test on an implementation of the ReLU function using a look-up table.

Figure 8.15 – Comparison between the t-test on the state of the neuron between a basic implementation and our implementation with a look-up table.

As we can observe in Figure 8.15, our countermeasure does not completely remove the leakage associated with the sign of the activation value, but rather diminishes the amplitude of the t-test result by a factor of 3. These results suggest that this countermeasure alone is insufficient. However, the look-up table can be easily implemented alongside other countermeasures such as shuffling.

8.3.2 Shuffling the Neurons

We now propose to adapt a countermeasure that has already been used to secure cryptographic implementations: shuffling (see Subsection 3.2.4). The idea is to randomize the order of the neurons in the for loop of the ReLU function at each execution of this function. Indeed, in all the attacks presented in this thesis, the attacker supposes that she knows which neuron is being processed at a given time. By randomizing the order at each call of this function, the attacker can no longer determine which neurons correspond to which part of the side-channel traces. We present the pseudo-code implementation of our countermeasure in Algorithm 6.

Similarly to Subsection 8.3.1, we validate our countermeasure by computing a t-test

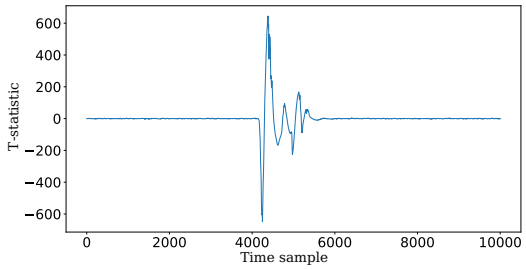
Algorithm 6 Pseudo-code of the shuffled ReLU function.

Require: A vector of $n \in \mathbb{N}$ pre-activation values $\boldsymbol{Z} \in \mathbb{R}^n$, $\boldsymbol{I} \in \mathbb{N}^n$ a vector of indexes such that $\boldsymbol{I}_i = i$, $\forall i \in [\![0; n]\!]$.
Ensure: $A = \texttt{max}(0, \boldsymbol{Z}) \in \mathbb{R}^n$
  for $index \leftarrow 0$ to n **do**
    $j \leftarrow \texttt{random}\ \%\ (index + 1)$ ▷ Generate a random integer modulo index + 1
    $tmp \leftarrow \boldsymbol{I}_j$
    $\boldsymbol{I}_j \leftarrow \boldsymbol{I}_{index}$
    $\boldsymbol{I}_{index} \leftarrow tmp$
  end for
  for $index \leftarrow 0$ to n **do**
    $i \leftarrow \boldsymbol{I}_{index}$
    $\boldsymbol{A_i} \leftarrow (\sim (\boldsymbol{Z_i} \gg 7))\ \&\ \boldsymbol{Z_i}$
  end for

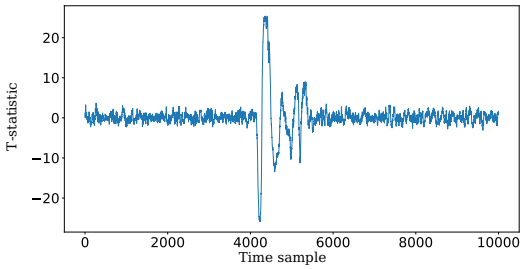
on the state of the activation value for different layer sizes. We target the same first neuron within the layer for each experiment. We collect $100,000$ side-channel traces in the configuration described in Subsection 8.2.2 and present the results in Figure 8.16.

We observe that the result of the t-test is significantly reduced with the countermeasure compared to without it. This result was expected, as with the countermeasure, the side-channel traces no longer always correspond to the targeted neuron. And as such, the shuffling makes the correlation between the state of the neuron and the side-channel traces less visible. However, it does not remove this correlation, and an attacker could realign the neuron or acquire a larger number of traces and filter those that do not correspond to the targeted neuron.

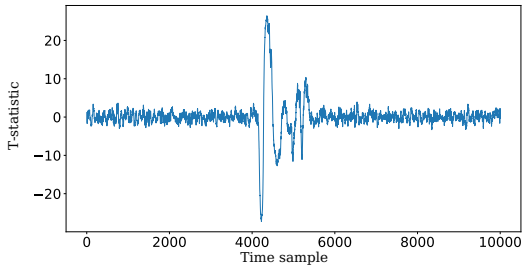
The difficulty of this process and the number of supplementary traces needed depend on the number of neurons within the layer. We can observe this pattern by comparing the maximum value of the t-test for a layer with 2 neurons, as represented in Figure 8.16a, compared to a layer with 64 neurons (see Figure 8.16d). Therefore, while this countermeasure increases the number of queries necessary to perform the attack, the attack remains possible. To further improve the security of activation functions against side-channel attacks, developers should consider a masked implementation of the activation function, as described in [DCA19]. We leave the improvement of the protection of activation functions via masking for future work.



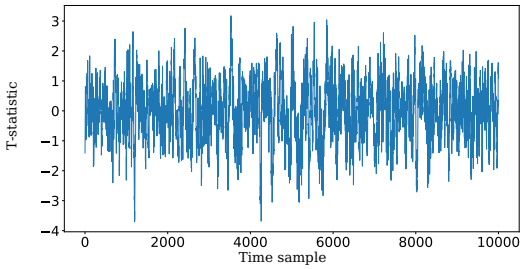
(a) T-test with shuffling on the the first layer with 2 neurons.



(b) T-test with shuffling on the the first layer with 8 neurons.



(c) T-test with shuffling on the the first layer with 32 neurons.



(d) T-test with shuffling on the the first layer with 64 neurons.

Figure 8.16 – Evolution of the t-statistic on the activity of the neuron for different layer sizes with our shuffling countermeasure.

8.4 Conclusion

In this chapter, we describe how an attacker can use cryptanalytical extraction to target the first layer of DNNs and then leverage this layer to profile the ReLU function, enabling an attack on the remaining layers. This solution naturally removes several restrictions present in previous fidelity-based model extraction attacks at the weight level. In particular, this methodology enables the extraction of quantized DNNs without being impacted by the depth of the target. We attempt to demonstrate our methodology on quantized DNNs on an MCU, but are unsuccessful in replicating the profiled side-channel attack on different neurons. However, we prove the viability of our methodology by simulating the side-channel attack and extracting complete quantized MLPs without any errors in the weights. Finally, we propose two countermeasures based on a new LUT implementation of the function and shuffling to block the first step of our methodology.

A large number of future works can be considered to improve the attack presented

in this chapter. The most important one is to successfully execute a side-channel attack to extract the activation values of all neurons. A possibility would be to depackage the chip of the targeted MCU and attempt the attack again. Another consideration is the assumption made for the profiling attack. In particular, due to the failure of the side-channel attack, we are unable to verify our methodology for extracting the scaling factor. Moreover, to avoid issues with unbalanced datasets, we choose to target DNNs without a shift factor to avoid saturation of the activation value. We argue that future work should consider this restriction and include attacks with good performance even with unbalanced datasets. Furthermore, while we demonstrated our methodology on MLPs, there are no restrictions on the architecture of the target. As such, our work could be extended to CNNs and more complex architectures. Finally, one of the limitations of our attack is the large computational complexity. As such, we argue that future works should improve the extraction process by leveraging the potential selection of equations to reduce the number of iterations of the RANSAC algorithm.

Conclusion

In this thesis, we focus on the security of AI and, more specifically, of DL models through the lens of SCA. We first studied how SCA could be used to improve already existing attacks against DNNs, and then demonstrated that SCA can also introduce completely new attack paths on these models. In this conclusion, we first summarize the contributions presented in this thesis. Finally, we discuss some potential perspectives and future works.

Summary of Contributions

Evasion attack is one of the most common threats faced by DNNs. However, a large number of attack frameworks are based on gradients, whether the true ones or estimated ones from the logits or the soft labels. In a black box scenario, an attacker has no access to either of these values and, as such, is unable to estimate the gradients. In this context, we introduce a profiled side-channel attack of the softmax function that enables an attacker to estimate the logits of the targeted DNN from a set of attack traces. In Chapter 5, we use DLSCAs to extract all logits of a DenseNet classifier with fewer than 10 attack traces and propose a complete framework for adversarial example generation, integrating this attack. While designed in the context of evasion attacks, we observe two results from this first contribution. The first one is that SCA can be used to extract private information that is useful in a large variety of attacks with different objectives. For example, the logits here can be used to improve evasion, as well as model extraction attacks. The second result is that the activation functions of DNNs are particularly interesting targets due to their placement within DNNs and the fact that most modern architectures are based on the same activation functions.

Based on these two observations, we propose a new methodology for fidelity-based model extraction attacks in Chapter 6. This methodology is inspired by cryptanalytical extraction but uses SCA as an oracle function to subdivide the targeted DNN at each activation function. With this method, we were able to target more complex and larger

architectures, such as a truncated MobileNetv1 composed of 11 layers and an MLP with 1.7 million parameters, while achieving very high fidelity, *i.e.*, 93.2% for the MLP and 88.4% for the truncated MobileNetv1. Furthermore, by introducing a methodology based on a divide-and-conquer approach, we remove several issues that cryptanalytical extraction frameworks had, such as the restriction on the type of layers and the robustness to special cases of neurons. Finally, we validate our contribution on all architectures extracted in the SOTA using cryptanalytical extraction, and outperform all methods in terms of maximum error on the weights.

While our approach to fidelity-based model extraction removes several issues, it does not solve all of them. In particular, quantization of the targeted DNN seems to drastically reduce the performance of both our framework and cryptanalytical extraction frameworks. For this reason, we decided to focus on attacks solely based on SCA to perform fidelity-based model extraction of quantized DNNs.

We first investigate the reason for the poor performance of cryptanalytical extraction when applied to quantized models. In Chapter 7, we linked this phenomenon to the quantization of inputs, which deteriorates the estimation of the critical points in cryptanalytical extraction. To overcome this issue, we introduced an attack based on supervised learning and SCA of the neuron's state. This method operates at the neuron level by introducing a penalty for models with a different side-channel behavior from the target. Using this approach, we were able to extract modern architectures, such as ResNet-18, ResNet-34, as well as VGG11, with a minimum fidelity of 90.16%.

However, despite its good performance, this method does not yield an exact copy of the targeted model, unlike cryptanalytical extraction. Therefore, in Chapter 8, we propose a methodology based on equations obtained from a profiled side-channel attack to extract the parameters of DNNs. This method is not impacted by quantization or propagation error and does not require a profiling device. As such, the threat model considered remains the same. Unfortunately, we were unable to perform the attack on the entire model. However, we simulated our complete methodology to evaluate its viability and showed that, despite significant query complexity, this method achieves perfect extraction of quantized DNNs.

Research Perspectives

The field of research on attacks against DL models is very active due to the interest in the potential application of these algorithms. However, we argue that for proper deployment of secured DNNs, research is still needed. In particular, in light of this thesis, we would like to highlight some potential research perspectives in the context of physical attacks.

Perspectives on evasion attacks. In our work, we propose to use a profiled side-channel attack to extract the logits of the targeted DNN. However, this attack requires both a profiling device and a large number of profiling traces. Finally, we only demonstrated its practicability on an MCU, which does not include parallelization of the operations. As such, we believe that future work could expand this attack in two directions. The first one is demonstrating its viability on more complex hardware architectures, such as TPUs or FPGAs, which are increasingly used for DNNs and also subject to side-channel attacks [Kur+24; Gon+24; Lom+25]. The second possibility is to improve the performance of our attack by reducing the number of profiling traces, or to search for a new side-channel attack that does not require a profiling device.

Another possibility is to adapt our work on the ReLU function and the side-channel attack to extract the state of the neurons and introduce a new method for generating adversarial examples. In particular, it should be possible to search, for example, by maximizing the difference in the internal states of the neurons, while minimizing the distance between the original and perturbed image.

Theorectical perspectives on model extraction. For fidelity-based model extraction, several perspectives can be highlighted that directly build upon existing work and our own. The first one is a direct adaptation of our methodology to other types of layers. We believe that such adaptations would increase the generality of our method, and as such, strengthen its applicability.

The second one regards the extraction of the last layer. While a first methodology achieving perfect fidelity has been recently proposed, we argue that achieving perfect fidelity is not sufficient in comparison with the high cost of fidelity-based model extraction at the weight level. Therefore, we argue that methods operating at the weight level should lead to a complete extraction of the targeted model and a complete copy. This implies

that further research is needed on the extraction of the last layer.

Furthermore, for extraction at the neuron level, aiming for perfect fidelity at a lower cost than extraction at the weight level, we believe that a theoretical analysis of the error in the weights and its impact on the fidelity of the extracted model would strengthen and ease future work.

Finally, although our research on this subject has been inconclusive, we believe that a connection between the extraction of quantized models with input quantization and the closest vector problem in a lattice can be made and exploited to provide theoretical boundaries on the error in the estimated weights. We believe that such theoretical guarantees would help to identify the threshold below which cryptanalytical methods no longer operate.

Practical perspectives on model extraction. The most direct future work for model extraction that we identify is to propose a successful profiled side-channel attack on the ReLU function to extract the activation values. Once this attack has been demonstrated on an MCU, the methodology introduced in Chapter 8 becomes operational.

Since none of our methodologies are restricted to ReLU-based DNNs, we believe it is necessary to adapt both side-channel attacks, *i.e.*, on the states and the activation values, to other activation functions as in [Che+25b]. Although ReLU remains the most common activation function, adapting our attack to other activation functions would make our framework more generic and agnostic of the model's architecture.

Finally, the contributions made in Chapter 6, Chapter 7, and Chapter 8 were made with the objective of targeting complex and modern architectures of DNNs. However, such models are now being deployed on parallelized hardware architectures. While we design our attacks with these targets in mind, we do not demonstrate the attacks on such hardware architectures.

Finally, while this thesis focuses on attacks against DNNs, the security of these models raises concerns from both application and data confidentiality perspectives. As such, we believe that a proper methodology for evaluating DL algorithms should be proposed and standardized, similarly to what is done with cryptographic systems [1].

1. https://www.commoncriteriaportal.org/files/ccfiles/CEM2022R1.pdf

List of publications & Communications

This thesis resulted in one publication in an international conference:

- Benoît Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, « When Side-Channel Attacks Break the Black-Box Property of Embedded Artificial Intelligence », *in*: *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023

Chapter 6 and Chapter 7 have also led to the following preprints, which will be submitted for review:

- Benoit Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, « A Hard-Label Cryptanalytic Extraction of Non-Fully Connected Deep Neural Networks using Side-Channel Attacks », *in*: *arXiv preprint arXiv:2411.10174* (2024)
- Benoit Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, « Evaluating the impact of quantization on Cryptanalytic Extraction of Deep Neural Networks », *in*: 2025

Parts of this research have also been presented on various occasions:

- Workshop on Practical Hardware Innovations in Security Implementation and Characterization (PHISIC) in Gardanne (2025)
- SemSecuElec Seminary in Rennes (2025)
- 1st European Symposium on Security and Artificial Intelligence (ESSAI) in Rennes (2024)
- Conférence Interne Thales (CONFIT) in Toulouse (2024)

BIBLIOGRAPHY

[Aba+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, « TensorFlow: A System for Large-Scale Machine Learning », *in*: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, Savannah, GA, USA: USENIX Association, 2016, pp. 265–283.

[ACW18] Anish Athalye, Nicholas Carlini, and David A. Wagner, « Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples », *in*: *International Conference on Machine Learning*, 2018.

[Arc+06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater, « Template Attacks in Principal Subspaces », *in*: *Cryptographic Hardware and Embedded Systems - CHES 2006*, ed. by Louis Goubin and Mitsuru Matsui, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–14.

[B18] Wieland Brendel *, Jonas Rauber *, and Matthias Bethge, « Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models », *in*: *International Conference on Learning Representations*, 2018.

[Bar+20] Antonio Barbalau, Adrian Cosma, Radu Tudor Ionescu, and Marius Popescu, « Black-Box Ripper: Copying black-box models using generative evolutionary algorithms », *in*: 33 (2020), ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, pp. 20120–20129.

[Bat+19] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek, « CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel », *in*: *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX Association, Aug. 2019, pp. 515–532.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier, « Correlation Power Analysis with a Leakage Model », *in*: *Cryptographic Hardware and Embedded Systems - CHES 2004*, ed. by Marc Joye and Jean-Jacques Quisquater, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29.

[Ber21] Rémi Bernhard, « Intégrité des réseaux de neurones : défenses contre les exemples adverses et leur transférabilité », Theses, Université de Lyon, Nov. 2021.

[BGL09] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust, « Differential Cluster Analysis », *in*: *Cryptographic Hardware and Embedded Systems - CHES 2009*, ed. by Christophe Clavier and Kris Gaj, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 112–127.

[Big+13] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli, « Evasion Attacks against Machine Learning at Test Time », *in*: *Machine Learning and Knowledge Discovery in Databases*, ed. by Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 387–402.

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville, « Estimating or propagating gradients through stochastic neurons for conditional computation », *in*: *arXiv preprint arXiv:1308.3432* (2013).

[Blo+18] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'Brien, Yaman Umuroglu, Miriam Leeser, and Kees A. Vissers, « FINN-R », *in*: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11 (2018), pp. 1–23.

[BM24] Maor Biton Dor and Yisroel Mirsky, « Efficient Model Extraction via Boundary Sampling », *in*: *Proceedings of the 2024 Workshop on Artificial Intelligence and Security*, AISec '24, Salt Lake City, UT, USA: Association for Computing Machinery, 2024, pp. 1–11.

[BMD19] Rémi Bernhard, Pierre-Alain Moëllic, and Jean-Max Dutertre, « Impact of Low-Bitwidth Quantization on the Adversarial Robustness for Embedded Neural Networks », *in*: *2019 International Conference on Cyberworlds (CW)* (2019), pp. 308–315.

[BNS19] Ron Banner, Yury Nahshan, and Daniel Soudry, « Post training 4-bit quantization of convolutional networks for rapid-deployment », *in*: *Advances in neural information processing systems* 32 (2019).

[Bre+18] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu, « Practical Fault Attack on Deep Neural Networks », *in*: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, pp. 2204–2206.

[Bru+15] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, Damien Marion, and Olivier Rioul, « Less is More », *in*: *Cryptographic Hardware and Embedded Systems – CHES 2015*, ed. by Tim Güneysu and Helena Handschuh, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 22–41.

[Buz+23] Gon Buzaglo, Niv Haim, Gilad Yehudai, Gal Vardi, Yakir Oz, Yaniv Nikankin, and michal Irani, « Deconstructing Data Reconstruction: Multiclass, Weight Decay and General Losses », *in*: *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[BZR18] Saman Biookaghazadeh, Ming Zhao, and Fengbo Ren, « Are FPGAs Suitable for Edge Computing? », *in*: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA: USENIX Association, July 2018.

[Can+24] Isaac A. Canales-Martínez, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, Nitin Satpute, and Adi Shamir, « Polynomial Time Cryptanalytic Extraction of Neural Network Models », *in*: *Advances in Cryptology – EUROCRYPT 2024*, ed. by Marc Joye and Gregor Leander, Cham: Springer Nature Switzerland, 2024, pp. 3–33.

[Car+21] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, A. Terzis, and Florian Tramèr, « Membership Inference Attacks From First Principles », *in*: *2022 IEEE Symposium on Security and Privacy (SP)* (2021), pp. 1897–1914.

[Car+25] Nicholas Carlini, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, and Adi Shamir, « Polynomial Time Cryptanalytic Extraction of Deep Neural Networks in the Hard-Label Setting », *in*: *Advances in Cryptology – EUROCRYPT 2025*, ed. by Serge Fehr and Pierre-Alain Fouque, Cham: Springer Nature Switzerland, 2025, pp. 364–396.

[CAU47] A. CAUCHY, « Methode generale pour la resolution des systemes d'equations simultanees », *in*: *Comptes Rendus Hebdomadaire Seances Académie des sciences de Paris* 25 (1847), pp. 536–538.

[CBD15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, « BinaryConnect: Training Deep Neural Networks with binary weights during propagations », *in*: *Neural Information Processing Systems*, 2015.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous, « Differential Power Analysis in the Presence of Hardware Countermeasures », *in*: *Cryptographic Hardware and Embedded Systems — CHES 2000*, ed. by Çetin K. Koç and Christof Paar, Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 252–263.

[CCP17] Eleonora Cagli, Cécile Canovas, and Emmanuel Prouff, « Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing », *in*: *Workshop on Cryptographic Hardware and Embedded Systems*, 2017.

[CDG20] Hervé Chabanne, Vincent Despiegel, and Linda Guiga, « A protection against the extraction of neural network models », *in*: *arXiv preprint arXiv:2005.12782* (2020).

[CDP16] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff, « Enhancing Dimensionality Reduction Methods for Side-Channel Attacks », *in*: *Smart Card Research and Advanced Applications*, ed. by Naofumi Homma and Marcel Medwed, Cham: Springer International Publishing, 2016, pp. 15–33.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff, « Kernel Discriminant Analysis for Information Extraction in the Presence of Masking », *in*: *Smart Card Research and Advanced Applications*, ed. by Kerstin Lemke-Rust and Michael Tunstall, Cham: Springer International Publishing, 2017, pp. 1–22.

[Cha+99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi, « Towards Sound Approaches to Counteract Power-Analysis Attacks », *in*: *Advances in Cryptology — CRYPTO' 99*, ed. by Michael Wiener, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 398–412.

[Che+17a] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh, « ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models », *in*: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (2017).

[Che+17b] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song, « Targeted backdoor attacks on deep learning systems using data poisoning », *in*: *arXiv preprint arXiv:1712.05526* (2017).

[Che+18] Yukang Chen, Gaofeng Meng, Qian Zhang, Xinbang Zhang, Liangchen Song, Shiming Xiang, and Chunhong Pan, « Joint Neural Architecture Search and Quantization », *in*: *ArXiv* abs/1811.09426 (2018).

[Che+25a] Yi Chen, Xiaoyang Dong, Jian Guo, Yantian Shen, Anyu Wang, and Xiaoyun Wang, « Hard-Label Cryptanalytic Extraction of Neural Network Models », *in*: *Advances in Cryptology – ASIACRYPT 2024*, ed. by Kai-Min Chung and Yu Sasaki, Singapore: Springer Nature Singapore, 2025, pp. 207–236.

[Che+25b] Yi Chen, Xiaoyang Dong, Ruijie Ma, Yantian Shen, Anyu Wang, Hongbo Yu, and Xiaoyun Wang, « Delving into Cryptanalytic Extraction of PReLU Neural Networks », *in*: *arXiv preprint arXiv:2509.16620* (2025).

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov, « Cryptanalytic Extraction of Neural Network Models », *in*: *Advances in Cryptology – CRYPTO 2020*, ed. by Daniele Micciancio and Thomas Ristenpart, Cham: Springer International Publishing, 2020, pp. 189–218.

[CJW20] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright, « HopSkipJumpAttack: A Query-Efficient Decision-Based Attack », *in*: *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1277–1294.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov, « An Efficient Method for Random Delay Generation in Embedded Software », *in*: *Cryptographic Hardware and Embedded Systems - CHES 2009*, ed. by Christophe Clavier and Kris Gaj, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 156–170.

[CK15] Marios Choudary and Markus Kuhn, « Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits », *in*: *Smart Card Research and Advanced Applications*, ed. by Marc Joye and Amir Moradi, Cham: Springer International Publishing, 2015, pp. 85–103.

[Coq+23] Benoît Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, « When Side-Channel Attacks Break the Black-Box Property of Embedded Artificial Intelligence », *in*: *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023.

[Coq+24a] Benoit Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, « A Hard-Label Cryptanalytic Extraction of Non-Fully Connected Deep Neural Networks using Side-Channel Attacks », *in*: *arXiv preprint arXiv:2411.10174* (2024).

[Coq+24b] Benoît Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, « When Side-Channel Attacks Break the Black-Box Property of Embedded Artificial Intelligence », *in*: *1st European Symposium on Security and Artificial Intelligence (ESSAI), invited*, Rennes, France, 2024.

[Coq+25] Benoit Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid, « Evaluating the impact of quantization on Cryptanalytic Extraction of Deep Neural Networks », *in*: 2025.

[CRK19] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter, « Certified Adversarial Robustness via Randomized Smoothing », *in*: *Proceedings of the 36th International Conference on Machine Learning*, ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov, vol. 97, Proceedings of Machine Learning Research, PMLR, June 2019, pp. 1310–1320.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi, « Template Attacks », *in*: *Cryptographic Hardware and Embedded Systems - CHES 2002*, ed. by Burton S. Kaliski, çetin K. Koç, and Christof Paar, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 13–28.

[CS25] Isaac A. Canales-Martínez and David Santos, *Extracting Some Layers of Deep Neural Networks in the Hard-Label Setting*, Cryptology ePrint Archive, Paper 2025/1118, 2025.

[CV95] Corinna Cortes and Vladimir Vapnik, « Support-Vector Networks », *in*: *Mach. Learn.* 20.*3* (Sept. 1995), pp. 273–297.

[CW16] Nicholas Carlini and David A. Wagner, « Towards Evaluating the Robustness of Neural Networks », *in*: *2017 IEEE Symposium on Security and Privacy (SP)* (2016), pp. 39–57.

[Cyb89] George V. Cybenko, « Approximation by superpositions of a sigmoidal function », *in*: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314.

[CZ06] Zhimin Chen and Yujie Zhou, « Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage », *in*: *Cryptographic Hardware and Embedded Systems - CHES 2006*, ed. by Louis Goubin and Mitsuru Matsui, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 242–254.

[DCA19] Anuj Dubey, Rosario Cammarota, and Aydin Aysu, « MaskedNet: The First Hardware Inference Engine Aiming Power Side-Channel Protection », *in*: *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2019), pp. 197–208.

[DCA20] Anuj Dubey, Rosario Cammarota, and Aydin Aysu, « BoMaNet: Boolean Masking of an Entire Neural Network », *in*: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)* (2020), pp. 1–9.

[Den+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, « ImageNet: A large-scale hierarchical image database », *in*: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 248–255.

[Den12] Li Deng, « The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web] », *in*: *IEEE Signal Processing Magazine* 29.*6* (2012), pp. 141–142.

[DHG01] Richard Duda, Peter Hart, and David G.Stork, « Pattern Classification », *in*: Jan. 2001.

[FB81] Martin A. Fischler and Robert C. Bolles, « Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography », *in*: *Commun. ACM* 24.*6* (June 1981), pp. 381–395.

[Fis36] Rory A. Fisher, « THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS », *in*: *Annals of Human Genetics* 7 (1936), pp. 179–188.

[Foe+24] Hanna Foerster, Robert Mullins, Ilia Shumailov, and Jamie Hayes, « Beyond Slow Signs in High-fidelity Model Extraction », *in*: *Advances in Neural Information Processing Systems*, ed. by A. Globerson, L. Mackey, D. Belgrave,

A. Fan, U. Paquet, J. Tomczak, and C. Zhang, vol. 37, Curran Associates, Inc., 2024, pp. 19496–19522.

[FPF25] Giuseppe Franco, Alessandro Pappalardo, and Nicholas J Fraser, *Xilinx/brevitas*, 2025.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, `http://www.deeplearningbook.org`, MIT Press, 2016.

[Gie+08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel, « Mutual Information Analysis », *in*: *Cryptographic Hardware and Embedded Systems – CHES 2008*, ed. by Elisabeth Oswald and Pankaj Rohatgi, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 426–442.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier, « Electromagnetic Analysis: Concrete Results », *in*: *Cryptographic Hardware and Embedded Systems — CHES 2001*, ed. by Çetin K. Koç, David Naccache, and Christof Paar, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 251–261.

[Gon+24] Cheng Gongye, Yukui Luo, Xiaolin Xu, and Yunsi Fei, « Side-Channel-Assisted Reverse-Engineering of Encrypted DNN Hardware Accelerator IP and Attack Surface Exploration », *in*: *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4678–4695.

[Goo+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio, « Generative Adversarial Nets », *in*: *Neural Information Processing Systems*, 2014.

[GP99] Louis Goubin and Jacques Patarin, « DES and Differential Power Analysis The "Duplication" Method », *in*: *Cryptographic Hardware and Embedded Systems*, ed. by Çetin K. Koç and Christof Paar, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 158–172.

[GSS14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, « Explaining and Harnessing Adversarial Examples », *in*: *International Conference on Learning Representations* abs/1412.6572 (2014).

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer, « RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis », *in*: *Advances in Cryptology – CRYPTO 2014*, ed. by Juan A. Garay and Rosario Gennaro, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 444–461.

[Guo+18] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten, « Countering Adversarial Images using Input Transformations », *in*: *International Conference on Learning Representations*, 2018.

[Gup+15] Suyog Gupta, Ankur Agrawal, K. Gopalakrishnan, and Pritish Narayanan, « Deep Learning with Limited Numerical Precision », *in*: *International Conference on Machine Learning*, 2015.

[Hai+22] Niv Haim, Gal Vardi, Gilad Yehudai, michal Irani, and Ohad Shamir, « Reconstructing Training Data From Trained Neural Networks », *in*: *Advances in Neural Information Processing Systems*, ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, 2022.

[He+15a] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun, « Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification », *in*: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1026–1034.

[He+15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, « Deep Residual Learning for Image Recognition », *in*: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.

[Hec+23] Kévin Hector, Mathieu Dumont, Pierre-Alain Moellic, and Jean-Max Dutertre, « Fault injection and safe-error attack for extraction of embedded neural network models », *in*: *ESORICS 2023 International Workshops*, vol. 14399, Computer Security. ESORICS 2023 International Workshops CPS4CIP, ADIoT, SecAssure, WASP, TAURIN, PriST-AI, and SECAI, The Hague, The Netherlands, September 25–29, 2023, Revised Selected Papers, Part II, Delft University of Technology, La Hague, Netherlands, Sept. 2023, pp. 644–664.

[Hey+13] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl, « Clustering Algorithms for Non-profiled Single-Execution Attacks on Exponentiations », *in*: *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, Berlin, Germany: Springer-Verlag, 2013, pp. 79–93.

[HK70] Arthur E. Hoerl and Robert W. Kennard, « Ridge Regression: Biased Estimation for Nonorthogonal Problems », *in*: *Technometrics* 12.*1* (1970), pp. 55–67.

[HLW16] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger, « Densely Connected Convolutional Networks », *in*: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2261–2269.

[HMD15] Song Han, Huizi Mao, and William J Dally, « Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding », *in*: *arXiv preprint arXiv:1510.00149* (2015).

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard, « An AES Smart Card Implementation Resistant to Power Analysis Attacks », *in*: July 2006, pp. 239–252.

[Hor+25] Peter Horvath, Lukasz Chmielewski, Léo Weissbart, Lejla Batina, and Yuval Yarom, « {BarraCUDA}: Edge {GPUs} do Leak {DNN} Weights », *in*: *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 4017–4034.

[Hor14] Mark Horowitz, « 1.1 Computing's energy problem (and what we can do about it) », *in*: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (2014), pp. 10–14.

[How+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, « MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications », *in*: *ArXiv* abs/1704.04861 (2017).

[How+19] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam, « Searching for MobileNetV3 », *in*: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 1314–1324.

[HR19] Boris Hanin and David Rolnick, « Deep ReLU Networks Have Surprisingly Few Activation Patterns », *in*: *Neural Information Processing Systems*, 2019.

[HS14] Michael Hutter and Jörn-Marc Schmidt, « The Temperature Side Channel and Heating Fault Attacks », *in*: *Smart Card Research and Advanced Applications*, ed. by Aurélien Francillon and Pankaj Rohatgi, Cham: Springer International Publishing, 2014, pp. 219–235.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber, « Long Short-Term Memory », *in*: *Neural Computation* 9 (1997), pp. 1735–1780.

[HVD15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean, « Distilling the Knowledge in a Neural Network », *in*: *ArXiv* abs/1503.02531 (2015).

[IEM19] Andrew Ilyas, Logan Engstrom, and Aleksander Madry, « Prior Convictions: Black-Box Adversarial Attacks with Bandits and Priors », *in*: *International Conference on Learning Representations* (2019).

[Ily+18] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin, « Black-box Adversarial Attacks with Limited Queries and Information », *in*: *Proceedings of the 35th International Conference on Machine Learning*, ed. by Jennifer Dy and Andreas Krause, vol. 80, Proceedings of Machine Learning Research, PMLR, Oct. 2018, pp. 2137–2146.

[IPS02] J. Irwin, D. Page, and N.P. Smart, « Instruction stream mutation for non-deterministic processors », *in*: *Proceedings IEEE International Conference on Application- Specific Systems, Architectures, and Processors*, 2002, pp. 286–295.

[IS15] Sergey Ioffe and Christian Szegedy, « Batch normalization: Accelerating deep network training by reducing internal covariate shift », *in*: *International conference on machine learning*, pmlr, 2015, pp. 448–456.

[Jag+20] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot, « High Accuracy and High Fidelity Extraction of Neural Networks », *in*: *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, Aug. 2020, pp. 1345–1362.

[Jou+23a] Raphaël Joud, Pierre-Alain Moëllic, Simon Pontié, and Jean-Baptiste Rigaud, « A Practical Introduction to Side-Channel Extraction of Deep Neural Network Parameters », *in*: *Smart Card Research and Advanced Applications*, ed. by Ileana Buhan and Tobias Schneider, Cham: Springer International Publishing, 2023, pp. 45–65.

[Jou+23b] Raphaël Joud, Pierre-Alain Moëllic, Simon Pontié, and Jean-Baptiste Rigaud, « Like an Open Book? Read Neural Network Architecture with Simple Power Analysis on 32-Bit Microcontrollers », *in*: *Smart Card Research and Advanced Applications - 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14-16, 2023, Revised Selected Papers*, ed. by Shivam Bhasin and Thomas Roche, vol. 14530, Lecture Notes in Computer Science, Springer, 2023, pp. 256–276.

[KB15] Diederik Kingma and Jimmy Ba, « Adam: A Method for Stochastic Optimization », *in*: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio, « Adversarial Machine Learning at Scale », *in*: *International Conference on Learning Representations*, 2017.

[KGB18] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio, « Adversarial examples in the physical world », *in*: *Artificial intelligence safety and security*, Chapman and Hall/CRC, 2018, pp. 99–112.

[KH09] Alex Krizhevsky and Geoffrey Hinton, *Learning multiple layers of features from tiny images*, tech. rep. 0, Toronto, Ontario: University of Toronto, 2009.

[KHS21] Faiq Khalid, Muhammad Abdullah Hanif, and Muhammad Shafique, « Exploiting vulnerabilities in deep neural networks: Adversarial and fault-injection attacks », *in*: *arXiv preprint arXiv:2105.03251* (2021).

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun, « Differential Power Analysis », *in*: *Advances in Cryptology — CRYPTO' 99*, ed. by Michael Wiener, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[Koc96] Paul C. Kocher, « Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems », *in*: *Advances in Cryptology — CRYPTO '96*, ed. by Neal Koblitz, Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[KPQ20] Sanjay Kariyappa, Atul Prakash, and Moinuddin K. Qureshi, « MAZE: Data-Free Model Stealing Attack Using Zeroth-Order Gradient Estimation », *in*: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 13809–13818.

[Kri10] Alex Krizhevsky, « Convolutional Deep Belief Networks on CIFAR-10 », *in*: 2010.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, « ImageNet classification with deep convolutional neural networks », *in*: *Communications of the ACM* 60 (2012), pp. 84–90.

[Kur+24] Ashley Kurian, Anuj Dubey, Ferhat Yaman, and Aydin Aysu, « TPUXtract: An Exhaustive Hyperparameter Extraction Framework », *in*: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.*1* (Dec. 2024), pp. 78–103.

[LDS89] Yann LeCun, John S. Denker, and Sara A. Solla, « Optimal Brain Damage », *in*: *Neural Information Processing Systems*, 1989.

[Léc+18] Mathias Lécuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel J. Hsu, and Suman Sekhar Jana, « Certified Robustness to Adversarial Examples with Differential Privacy », *in*: *2019 IEEE Symposium on Security and Privacy (SP)* (2018), pp. 656–672.

[Lec+98] Yann. Lecun, Leon. Bottou, Yoshua. Bengio, and Patrick Haffner, « Gradient-based learning applied to document recognition », *in*: *Proceedings of the IEEE* 86.*11* (1998), pp. 2278–2324.

[Lec89] Yann Lecun, « Generalization and network design strategies », English (US), *in*: *Connectionism in perspective*, ed. by R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Elsevier, 1989.

[Li+20] Huichen Li, Xiaojun Xu, Xiaolu Zhang, Shuang Yang, and Bo Li, « QEBA: Query-Efficient Boundary-Based Blackbox Attack », *in*: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 1218–1227.

[Liu+17] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song, « Delving into Transferable Adversarial Examples and Black-box Attacks », *in*: *International Conference on Learning Representations*, 2017.

[Liu+25] Haolin Liu, Adrien Siproudhis, Samuel Experton, Peter Lorenz, Christina Boura, and Thomas Peyrin, « Navigating the Deep: Signature Extraction on Deep Neural Networks », *in*: *ArXiv* abs/2506.17047 (2025).

[LM05] Daniel Lowd and Christopher Meek, « Adversarial learning », *in*: *Knowledge Discovery and Data Mining*, 2005.

[Lom+25] Guillaume Lomet, Rubén Salvador, Brice Colombier, Vincent Grosso, Olivier Sentieys, and Cédric Killian, « Side-Channel Extraction of Dataflow AI Accelerator Hardware Parameters », *in*: *2025 IEEE 31st International Sympo-*

sium on On-Line Testing and Robust System Design (IOLTS), 2025, pp. 1–7.

[LSC18] Liangzhen Lai, Naveen Suda, and Vikas Chandra, « CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs », *in*: *ArXiv* abs/1801.06601 (2018).

[LTA18] Griffin Lacey, Graham W. Taylor, and Shawki M. Areibi, « Stochastic Layer-Wise Precision in Deep Neural Networks », *in*: *Conference on Uncertainty in Artificial Intelligence*, 2018.

[Lu20] Lu Lu, « Dying ReLU and Initialization: Theory and Numerical Examples », *in*: *Communications in Computational Physics* 28.*5* (June 2020), pp. 1671–1706, arXiv: `1903.06733 [stat.ML]`.

[Ma20] Jianjia Ma, *A higher-level Neural Network library on Microcontrollers (NNoM)*, version v0.4.2, Oct. 2020.

[Mad+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu, « Towards Deep Learning Models Resistant to Adversarial Attacks », *in*: *International Conference on Learning Representations*, 2018.

[Mag19] Houssem Maghrebi, *Deep Learning based Side Channel Attacks in Practice*, Cryptology ePrint Archive, Paper 2019/578, 2019.

[Man04] Stefan Mangard, « Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness », *in*: *Topics in Cryptology – CT-RSA 2004*, ed. by Tatsuaki Okamoto, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 222–235.

[Mar+23] Flavio Martinelli, Berfin Simsek, Wulfram Gerstner, and Johanni Brea, « Expand-and-Cluster: Parameter Recovery of Neural Networks », *in*: *International Conference on Machine Learning*, 2023.

[Mas+23] Loïc Masure, Gaëtan Cassiers, Julien Hendrickx, and François-Xavier Standaert, « Information Bounds and Convergence Rates for Side-Channel Security Evaluators », *in*: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.*3* (June 2023), pp. 522–569.

[MBC21] Saurav Maji, Utsav Banerjee, and Anantha P. Chandrakasan, « Leaky Nets: Recovering Embedded Neural Network Models and Inputs Through Simple Power and Timing Side-Channels—Attacks and Defenses », *in*: *IEEE Internet of Things Journal* 8.*15* (2021), pp. 12079–12092.

[Mel+19] Eldad Meller, Alexander Finkelstein, Uri Almog, and Mark Grobman, « Same, Same But Different: Recovering Neural Network Quantization Error Through Weight Factorization », *in*: *Proceedings of the 36th International Conference on Machine Learning*, ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov, vol. 97, Proceedings of Machine Learning Research, PMLR, Sept. 2019, pp. 4486–4495.

[Mil+18] Smitha Milli, Ludwig Schmidt, Anca D. Dragan, and Moritz Hardt, « Model Reconstruction from Model Explanations », *in*: *Proceedings of the Conference on Fairness, Accountability, and Transparency* (2018).

[Mon+14] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio, « On the Number of Linear Regions of Deep Neural Networks », *in*: *Neural Information Processing Systems*, 2014.

[Moo+03] Simon Moore, Ross Anderson, Robert Mullins, George Taylor, and Jacques J.A. Fournier, « Balanced self-checking asynchronous logic for smart card applications », *in*: *Microprocessors and Microsystems* 27.*9* (2003), pp. 421–430.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, Jan. 2007.

[Nag+21] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort, « A White Paper on Neural Network Quantization », *in*: *ArXiv* abs/2106.08295 (2021).

[NC18] Erick Nascimento and Łukasz Chmielewski, « Applying Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations », *in*: *Smart Card Research and Advanced Applications*, ed. by Thomas Eisenbarth and Yannick Teglia, Cham: Springer International Publishing, 2018, pp. 213–231.

[NH10] Vinod Nair and Geoffrey E. Hinton, « Rectified Linear Units Improve Restricted Boltzmann Machines », *in*: *International Conference on Machine Learning*, 2010.

[NSF21] Tsunato Nakai, Daisuke Suzuki, and Takeshi Fujino, « Timing Black-Box Attacks: Crafting Adversarial Examples through Timing Leaks against DNNs on Embedded Devices », *in*: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.*3* (July 2021), pp. 149–175.

[OMR23] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber, « I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences », *in*: *ACM Computing Surveys* 55.*14s* (July 2023), pp. 1–41.

[OSF18] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz, « Knockoff Nets: Stealing Functionality of Black-Box Models », *in*: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018), pp. 4949–4958.

[Pap+15] Nicolas Papernot, Patrick Mcdaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami, « The Limitations of Deep Learning in Adversarial Settings », *in*: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2015), pp. 372–387.

[Pap+16] Nicolas Papernot, Patrick Mcdaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami, « Practical Black-Box Attacks against Machine Learning », *in*: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (2016).

[Pap+18] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long, « Technical Report on the CleverHans v2.1.0 Adversarial Examples Library », *in*: *arXiv preprint arXiv:1610.00768* (2018).

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, « PyTorch: An Imperative Style, High-Performance Deep Learning Library », *in*: *Proceedings of the 33rd Inter-*

national Conference on Neural Information Processing Systems, Red Hook, NY, USA: Curran Associates Inc., 2019.

[Per+20] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek, « Keep it Unsupervised: Horizontal Attacks Meet Deep Learning », *in*: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.*1* (Dec. 2020), pp. 343–372.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow, « Transferability in machine learning: from phenomena to black-box attacks using adversarial samples », *in*: *arXiv preprint arXiv:1605.07277* (2016).

[Pre12] Lutz Prechelt, « Early Stopping — But When? », *in*: *Neural Networks: Tricks of the Trade: Second Edition*, ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67.

[Qin+24] Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby R. Banbury, Chengxi Ye, Berkin Akin, Vaibhav Aggarwal, Tenghui Zhu, Daniele Moro, and Andrew Howard, « MobileNetV4 - Universal Models for the Mobile Ecosystem », *in*: *European Conference on Computer Vision*, 2024.

[Rak+21] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan, « DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories », *in*: *2022 IEEE Symposium on Security and Privacy (SP)* (2021), pp. 1157–1174.

[RGC15] Mauro Ribeiro, Katarina Grolinger, and Miriam A. M. Capretz, « MLaaS: Machine Learning as a Service », *in*: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)* (2015), pp. 896–902.

[RHF19] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan, « Bit-Flip Attack: Crushing Neural Network With Progressive Bit Search », *in*: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 1211–1220.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, « Learning representations by back-propagating errors », *in*: *Nature* 323 (1986), pp. 533–536.

[RK19] David Rolnick and Konrad Paul Kording, « Reverse-engineering deep ReLU networks », *in*: *International Conference on Machine Learning*, 2019.

[RO05] Christian Rechberger and Elisabeth Oswald, « Practical Template Attacks », *in*: *Information Security Applications*, ed. by Chae Hoon Lim and Moti Yung, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 440–456.

[Ros58] Frank Rosenblatt, « The perceptron: a probabilistic model for information storage and organization in the brain. », *in*: *Psychological review* 65.*6* (1958), p. 386.

[Rus+23] Manuele Rusci, Marco Fariselli, Martin Croome, Francesco Paci, and Eric Flamand, « Accelerating RNN-Based Speech Enhancement on a Multi-core MCU with Mixed FP16-INT8 Post-training Quantization », *in*: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ed. by Irena Koprinska, Paolo Mignone, Riccardo Guidotti, Szymon Jaroszewicz, Holger Fröning, Francesco Gullo, Pedro M. Ferreira, Damian Roqueiro, Gaia Ceddia, Slawomir Nowaczyk, João Gama, Rita Ribeiro, Ricard Gavaldà, Elio Masciari, Zbigniew Ras, Ettore Ritacco, Francesca Naretto, Andreas Theissler, Przemyslaw Biecek, Wouter Verbeke, Gregor Schiele, Franz Pernkopf, Michaela Blott, Ilaria Bordino, Ivan Luciano Danesi, Giovanni Ponti, Lorenzo Severini, Annalisa Appice, Giuseppina Andresini, Ibéria Medeiros, Guilherme Graça, Lee Cooper, Naghmeh Ghazaleh, Jonas Richiardi, Diego Saldana, Konstantinos Sechidis, Arif Canakoglu, Sara Pido, Pietro Pinoli, Albert Bifet, and Sepideh Pashami, Cham: Springer Nature Switzerland, 2023, pp. 606–617.

[SA08] François-Xavier Standaert and Cédric Archambeau, « Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages », *in*: *Cryptographic Hardware and Embedded Systems – CHES 2008*, ed. by Elisabeth Oswald and Pankaj Rohatgi, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 411–425.

[San+18] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, « MobileNetV2: Inverted Residuals and Linear Bottlenecks », *in*: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 4510–4520.

[Sho+16] R. Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov, « Membership Inference Attacks Against Machine Learning Models », *in*: *2017 IEEE Symposium on Security and Privacy (SP)* (2016), pp. 3–18.

[Shu+21] Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson, « Sponge Examples: Energy-Latency Attacks on Neural Networks », *in*: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021, pp. 212–231.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar, « A Stochastic Model for Differential Side Channel Cryptanalysis », *in*: *Cryptographic Hardware and Embedded Systems – CHES 2005*, ed. by Josyula R. Rao and Berk Sunar, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 30–46.

[Sri+14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, « Dropout: a simple way to prevent neural networks from overfitting », *in*: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[SRS17] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov, « Machine Learning Models that Remember Too Much », *in*: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017).

[SVS17] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai, « One Pixel Attack for Fooling Deep Neural Networks », *in*: *IEEE Transactions on Evolutionary Computation* 23 (2017), pp. 828–841.

[SZ14] Karen Simonyan and Andrew Zisserman, « Very Deep Convolutional Networks for Large-Scale Image Recognition », *in*: *International Conference on Learning Representations* abs/1409.1556 (2014).

[Sze+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, D. Erhan, Ian J. Goodfellow, and Rob Fergus, « Intriguing properties of neural networks », *in*: *International Conference on Learning Representations* abs/1312.6199 (2013).

[Sze+17] Vivienne Sze, Yu-hsin Chen, Tien-Ju Yang, and Joel S. Emer, « Efficient Processing of Deep Neural Networks: A Tutorial and Survey », *in*: *Proceedings of the IEEE* 105 (2017), pp. 2295–2329.

[Tak+22] Go Takatoi, Takeshi Sugawara, Kazuo Sakiyama, Yuko Hara-Azumi, and Y. Li, « The Limits of SEMA on Distinguishing Similar Activation Functions of Embedded Deep Neural Networks », *in*: *Applied Sciences* (2022).

[Tib11] Robert Tibshirani, « Regression shrinkage selection via the LASSO », *in*: *Journal of the Royal Statistical Society Series B* 73 (June 2011), pp. 273–282.

[Tim19] Benjamin Timon, « Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis », *in*: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.*2* (Feb. 2019), pp. 107–131.

[Tra+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart, « Stealing Machine Learning Models via Prediction APIs », *in*: *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX: USENIX Association, Aug. 2016, pp. 601–618.

[Tru+20] Jean-Baptiste Truong, Pratyush Maini, Robert J. Walls, and Nicolas Papernot, « Data-Free Model Extraction », *in*: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 4769–4778.

[Tu+18] Chun-Chen Tu, Pai-Shun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng, « AutoZOOM: Autoencoder-based Zeroth Order Optimization Method for Attacking Black-box Neural Networks », *in*: *AAAI Conference on Artificial Intelligence*, 2018.

[Ues+18] Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aaron van den Oord, « Adversarial Risk and the Dangers of Evaluating Against Weak Attacks », *in*: *Proceedings of the 35th International Conference on Machine Learning*, ed. by Jennifer Dy and Andreas Krause, vol. 80, Proceedings of Machine Learning Research, PMLR, Oct. 2018, pp. 5025–5034.

[Umu+16] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Heng Wai Leong, Magnus Jahre, and Kees A. Vissers, « FINN: A Framework for Fast, Scalable Binarized Neural Network Inference », *in*: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2016).

[Vas+17] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, « Attention is All you Need », *in*: *Neural Information Processing Systems*, 2017.

[Wei+18] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu, « 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, San Juan, PR, USA: Association for Computing Machinery, 2018, pp. 393–406.

[Xie+18] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille, « Mitigating Adversarial Effects Through Randomization », *in*: *International Conference on Learning Representations*, 2018.

[Yan+22] Xiao Yang, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu, « Boosting Transferability of Targeted Adversarial Examples via Hierarchical Generative Networks », *in*: *Computer Vision – ECCV 2022*, ed. by Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, Cham: Springer Nature Switzerland, 2022, pp. 725–742.

[Yos+14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson, « How transferable are features in deep neural networks? », *in*: *Advances in Neural Information Processing Systems*, ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, vol. 27, Curran Associates, Inc., 2014.

[ZDF23] Xiang Zhang, Aidong Adam Ding, and Yunsi Fei, « Deep-Learning Model Extraction Through Software-Based Power Side-Channel », *in*: *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)* (2023), pp. 1–9.

[Zha+22] Tianming Zhao, Yucheng Xie, Yan Wang, Jerry Cheng, Xiaonan Guo, Bin Hu, and Yingying Chen, « A Survey of Deep Learning on Mobile Devices: Applications, Optimizations, Challenges, and Research Opportunities », *in*: *Proceedings of the IEEE* 110.*3* (2022), pp. 334–354.

[Zha+25] Pinlong Zhao, Weiyao Zhu, Pengfei Jiao, Di Gao, and Ou Wu, « Data poisoning in deep learning: A survey », *in*: *arXiv preprint arXiv:2503.22759* (2025).

[ZL17] Barret Zoph and Quoc Le, « Neural Architecture Search with Reinforcement Learning », *in*: *International Conference on Learning Representations*, 2017.

[ZR20] Raphael Zingg and Matthias Rosenthal, « Artificial intelligence on microcontrollers », en, *in*: Embedded World Conference 2020, Nürnberg, 25.-27. Februar 2020, Feb. 2020.

[ZYW24] Zeliang Zhang, Wei Yao, and Xiaosen Wang, « Bag of tricks to boost adversarial transferability », *in*: *arXiv preprint arXiv:2401.08734* (2024).

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Etude de la Complémentarité des Attaques Logicielles et par Canaux Auxilaires contre des Algorithmes basé sur l'Apprentissage Profond

Mot clés : Réseaux Neuronaux Profonds, Attaque par Canal Auxiliaire, Extraction de modèle, Attaque par Evasion

Résumé : Les modèles de réseaux neuronaux profonds sont de plus en plus populaires, et les meilleurs modèles constituent désormais une propriété intellectuelle précieuse. Cependant, leur grande valeur et la surface d'attaque accrue causée par leur déploiement sur des systèmes embarqués ont conduit les attaquants à envisager de nouveaux types d'attaques, telles que les attaques par canaux auxiliaires. En conséquence, de nouveaux chemins d'attaque continuent d'apparaître, limitant la capacité des défenseurs à définir des contremesures efficaces. Pour limiter ces problèmes, cette thèse propose de se concentrer sur de nouvelles méthodologies d'attaque afin de démontrer l'impact des attaques par canal auxiliaire et pourquoi elles devraient être considérées. Cette thèse peut être subdivisée en deux parties. La première concerne l'amélioration des attaques existantes contre les algorithmes d'apprentissage profond à l'aide d'attaques par canal auxiliaire. Nous améliorons la faisabilité des attaques par évasion et l'extraction de modèles en ciblant deux fonctions d'activation populaires, les fonctions softmax et ReLU. Dans la deuxième partie de cette thèse, nous étudions l'impact de la quantification sur l'extraction de modèles et démontrons la nécessité d'introduire de nouvelles voies d'attaque basées sur les attaques par canal auxiliaire. Enfin, nous décrivons comment l'extraction de modèles basée sur la cryptanalyse ne permet pas extraire les DNNs quantifiés et présentons deux nouvelles attaques d'extraction de modèles basées sur la fidélité utilisant les canaux auxiliaires.

Title: On the Complementarity of Software and Side-Channel Attacks against Deep Learning Algorithms

Keywords: Deep Neural Networks, Side-Channel Attacks, Model Extraction, Evasion Attack

Abstract: Deep Neural Networks (DNNs) are becoming increasingly popular, and state-of-the-art models are now valuable intellectual property. However, the combination of their high value and increased attack surface caused by their deployment at the edge, has led attackers to consider new types of attack, such as side-channel attacks. As such, new attack paths continue to emerge, limiting the ability of defenders to define a security perimeter. To mitigate these issues, this thesis proposes focusing on new attack methodologies to demonstrate the impact of side-channel attacks and why developers should consider them. This thesis can be subdivided into two main studies. The first one is related to improving already existing attacks against DNNs using side-channel attacks. We enhance the feasibility of evasion and model extraction attacks by targeting two

popular activation functions, the softmax and the ReLU functions. In the second part of this thesis, we investigate the impact of quantization on model extractions and demonstrate the need to introduce new attack paths based on side-channel attacks. Finally, we describe how model extraction based on cryptanalysis failed to extract quantized DNNs and present two new fidelity-based model extraction attacks using side-channel analysis.