UNIVERSITÀ DEGLI STUDI DI BERGAMO

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
&
SCUOLA DI ALTA FORMAZIONE DOTTORALE

TESI DI DOTTORATO IN
TECHNOLOGY, INNOVATION AND MANAGEMENT

Secure and Efficient Deep Learning: From Adversarial Attacks to Robustness in Approximate Computing

Supervisore
Chiar.ma Prof.ssa
Valentina Casola

Candidato
Salvatore Della Torca
Matr. 1086465
XXXVIII ciclo

Anno Accademico 2024/2025

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
&
SCUOLA DI ALTA FORMAZIONE DOTTORALE

TESI DI DOTTORATO IN
TECHNOLOGY, INNOVATION AND MANAGEMENT

Secure and Efficient Deep Learning: From Adversarial Attacks to Robustness in Approximate Computing

Supervisore
Chiar.ma Prof.ssa
Valentina Casola

Candidato
Salvatore Della Torca
Matr. 1086465
XXXVIII ciclo

Anno Accademico 2024/2025

To those who truly love me
To those I truly love...

List of Publications

The work presented in this dissertation is the result of the research carried out during my Ph.D. studies. Below is the list of publications produced during this period, reported in chronological order. Please note that the first two and the last works are not directly related to the core contributions of this thesis, as they address different research topics. Finally, the last two papers are still under peer review at the time of writing.

1. Mario Barbareschi, Salvatore Barone, Valentina Casola, Salvatore Della Torca, and Daniele Lombardi. 2023. **Automatic Test Generation to Improve Scrum for Safety Agile Methodology**. *In Proceedings of the 18th International Conference on Availability, Reliability and Security (ARES '23)*. Association for Computing Machinery, New York, NY, USA, Article 137, 1–6.

2. S. Barone, V. Casola, S. D. Torca and D. Lombardi, **Timing Behavior Characterization of Critical Real-Time Systems through Hybrid Timing Analysis**, *2023 7th International Conference on System Reliability and Safety (ICSRS)*, Bologna, Italy, 2023, pp. 306-311.

3. Ahmadilivani, Mohammad Hasan and Barbareschi, Mario and Barone, Salvatore and Bosio, Alberto and Daneshtalab, Masoud and Torca, Salvatore Della and Gavarini, Gabriele and Jenihhin, Maksim and Raik, Jaan and Ruospo, Annachiara and Sanchez, Ernesto and Taheri, Mahdi, **Special Session: Approximation and Fault Resiliency of DNN Accelerators**, *2023 IEEE 41st VLSI Test Symposium (VTS)*, San Diego, CA, USA, 2023, pp. 1-10.

4. Barbareschi, M., Barone, S., Casola, V., Della Torca, S. (2025). **Error Resiliency and Adversarial Robustness in Convolutional Neural Networks: An Empirical Analysis**. In: Rey, G., Tigli, JY., Franquet, E. (eds) *Internet of Things. IFIPIoT 2024. IFIP Advances in Information and Communication Technology*, vol 737. Springer.

5. Valentina Casola and Salvatore Della Torca. 2024. **Transfer Adversarial Attacks through Approximate Computing**. *In Proceedings of the 19th International Conference on Availability, Reliability and Security (ARES '24)*. Association for Computing Machinery, New York, NY, USA, Article 168, 1–6.

6. S. Barone, V. Casola and S. Della Torca, **Ineffectiveness of Digital Transformations for Detecting Adversarial Attacks Against Quantized and Approximate CNNs**, *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, London, United Kingdom, 2024, pp. 290-295.

7. Salvatore Della Torca, Valentina Casola, and Simone Izzo. 2025. **N-Pixels: a Novel Grey-Box Adversarial Attack for Fooling Convolutional Neural Networks**. *In Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25)*.Association for Computing Machinery, New York, NY, USA, 1539–1547.

8. Salvatore Della Torca, Valentina Casola. **Incremental Pixels: a Black Box Adversarial Attack**. *Neural Networks*. (Currently under review, stage two).

9. Salvatore Della Torca, Antonio Bocchetti, Valentina Casola, Stefano Violante, **AI4AIDataSec: Improving the Security of AI-Generated Software with AI-Generated Secure Dataset**, *ACM Transactions on Software Engineering and Methodology*. (Currently under review).

List of Acronyms

The following acronyms are used throughout this text.

ADDNP Adversarial Detection method via Disentangling Natural images and Perturbations

AI Artificial Intelligence

ANN Artificial Neural Network

ARD Adversarially Robust Distillation

ART Adversarial Robustness ToolBox

AxC Approximate Computing

AxNN Approximate Neural Network

BERT Bidirectional Encoder Representations from Transformers

BIM Basic Iterative Method

BSR Block Shuffle and Rotation

CAG Content-aware Adversarial Attack Generator

CNN Convolutional Neural Network

CPU Central Processing Unit

CW Carlini-Wagner

DE Differential Evolution

DL Deep Learning

DNN Deep Neural Network

DSE Design Space Exploration

EAD Elastic-Net Attack against DNNs

ETA Ensemble Adversarial Training

FIA Feature Importance-aware Attack

FpNN Full-Precision Neural Network

FGSM Fast Gradient Sign Method

GAN Generative Adversarial Network

GPT Generative Pre-Trained Transformer

GP-GPU General Purpose - Graphic Processing Unit

GPU Graphic Processing Unit

HGD High-Level Representation Guided Denoising

IoT Internet of Things

IPs Incremental-Pixels

JSMA Jacobian-based Saliency Map Approach

KL Kullback-Leibler

LID Local Intrinsic Dimensionality

LLM Large Language Model

L-BFGS Limited-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm

ML Machine Learning

MI-FGSM Momentum Iterative FGSM

MMIA Mask Momentum Iterative Attack

MOO Multi-Objective Optimization

MOP Multi-objective Optimization Problem

MSE Mean Square Error

NLP Natural Language Processing

NN Neural Network

NSGA-II Non-dominated Sorting Genetic Algorithm-II

NPs N-Pixels

OP One-Pixel

PGD Projected Gradient Descent

PSNR Peak Signal-to-Noise Ratio

PTQ Post-Training Quantization

PX Pixle

QAT Quantization-Aware Training

QNN Quantized Neural Network

ReLU Rectified Linear Unit

SAP Stochastic Activation Pruning

SEM Scale Enriching Method

SGD Stochastic Gradient Descent

SimBA Simple Black-Box Adversarial Attack

SMBEA Serial-Mini-Batch-Ensemble-Attack

SotA State-of-the-Art

SQ Square Attack

SSIM Structural Similarity Index Measure

VAE Variational Autoencoder

WGAN Wasserstein GAN

ZOO Zeroth-Order Optimization

Table of contents

List of Publications **v**

List of Acronyms **ix**

Abstract **1**

1 Introduction **3**
1.1 Problem Statement and Contributions 6
1.1.1 Contributions of the Thesis 7
1.2 Structure of the Thesis 8

2 Related Work **11**
2.1 Basic Definitions and Machine Learning Concepts 12
2.1.1 Deep Learning and Deep Neural Networks 13
2.1.2 Generative Artificial Intelligence 17
2.2 Adversarial Threats and Defense in Machine Learning Systems 18
2.2.1 Symbols and Formal Definitions 20
2.2.2 Adversarial Threat Model 21
2.3 Evasion Attacks on CNNs 26

2.3.1 White-Box Attacks 27
2.3.2 Black-Box Attacks 32
2.3.3 Challenges 38
2.4 Adversarial Defense 40
2.4.1 Mitigation Defense-Methods 40
2.4.2 Detection Defense-Methods 45
2.4.3 Defense as Side Effect 47
2.4.4 Challenges 53

3 Improving the Stealthiness of Adversarial Attack against CNNs 55
3.1 N-Pixels: a Novel Grey-Box Adversarial Attack for Fooling Convolutional Neural Networks 58
3.1.1 Methodology 58
3.1.2 Experimental Evaluation 66
3.1.3 Discussion 75
3.2 Incremental Pixels: a Black Box Adversarial Attack 76
3.2.1 Methodology 76
3.2.2 Experimental Evaluation 84
3.2.3 Discussion 95

4 The Effects of Approximate Computing on Adversarial Machine Learning 97
4.1 Inspect-NN 98
4.1.1 Workflow 99
4.2 Experimental Testbed 104
4.3 Improving the Robustness of CNNs with Approximate Computing 109
4.3.1 Evaluation Metrics 111
4.3.2 Experimental Evaluation 112
4.3.3 Discussion 118
4.4 Transfer Adversarial Attacks trough Approximate Computing 120

4.4.1 Experimental Evaluation 125
4.4.2 Discussion 129
4.5 The Role of Approximate Computing on Defense 129
4.5.1 Experimental Evaluation 131
4.5.2 Discussion 134

5 Conclusions and Future Works **137**
5.1 Future Work 139
5.2 Closing Remarks 140

Bibliography **141**

List of figures **165**

List of tables **171**

Abstract

Deep Learning (DL) has achieved outstanding results in numerous domains, yet its widespread adoption in safety- and security-critical applications remains limited due to its vulnerability to adversarial examples. These imperceptible perturbations can drastically alter the predictions of Convolutional Neural Networks (CNNs), raising serious concerns about their trustworthiness.

This dissertation addresses the problem of adversarial robustness in CNNs from two complementary perspectives: the design of novel adversarial attacks and the evaluation of the effects of efficiency-oriented transformations on adversarial robustness.

First, we propose two new adversarial strategies specifically designed to balance effectiveness and stealthiness. N-Pixels is a gray-box attack that leverages internal representations of CNNs to identify the most relevant image regions and applies minimal perturbations for successful misclassification. Incremental Pixels is a black-box score-based attack that progressively perturbs pixels, increasing both the number and magnitude of modifications only when necessary. Both methods achieve a favorable trade-off between query efficiency, imperceptibility, and attack success rate.

Second, we investigate the interplay between adversarial robustness and efficiency-driven techniques, such as quantization and Approximate Computing. While these transformations are primarily adopted to reduce computational and energy costs, we show that they significantly influence adversarial behavior: in some cases mitigating the effect of specific attacks, in others introducing new weaknesses or reducing the reliability of defense mechanisms.

Overall, this thesis highlights the limitations of existing approaches and provides new insights into the relationship between efficiency, error resiliency, and adversarial robustness. The results contribute to a deeper understanding of how to design CNN-based systems that remain both efficient and secure in real-world scenarios.

1 Introduction

Over the last decade, Deep Learning (DL) has profoundly reshaped the landscape of Artificial Intelligence (AI). Thanks to its ability to learn complex representations directly from large amounts of data, DL has enabled machines to achieve, and in many cases surpass, human-level performance in a wide range of domains [1, 2]. Examples include (i) computer vision [3], where deep models now dominate tasks such as image classification, object detection, and semantic segmentation; (ii) speech processing [4], where they power State-of-the-Art (SotA) systems for speech recognition and speaker identification; (iii) Natural Language Processing (NLP) [5], where large-scale models achieve impressive results in translation, summarization, and dialogue systems; and (iv) and medical diagnostics [6], where CNN-based solutions are applied to radiology, histopathology, and genomics to support clinical decision making.

Within this technological revolution, Convolutional Neural Networks (CNNs) have emerged as one of the most promising and widely adopted architectures [7]. Their hierarchical structure, inspired by the visual cortex, enables them to automatically extract increasingly abstract features from raw data, reducing the need for manual feature engineering. This ability has made CNNs a cornerstone of modern AI research and applications. Today,

they form the backbone of numerous real-world systems: from autonomous vehicles [8], where they are used for scene understanding, traffic sign recognition, and pedestrian detection; to biometric authentication systems [9], where they enable robust face and fingerprint recognition; to smart surveillance solutions [10], where they are deployed for anomaly detection and activity monitoring; and even to industrial inspection [11], where they facilitate defect detection in manufacturing pipelines. The impact of CNNs extends beyond high-performance computing platforms. They have become central to many technologies embedded in consumer electronics—such as smartphones, smart assistants, and cameras—bringing DL capabilities into everyday life. In parallel, CNNs are increasingly deployed in mission- and safety-critical contexts, such as unmanned aerial vehicles, healthcare monitoring devices, and automotive driver-assistance systems. In all these domains, the combination of high accuracy, adaptability, and scalability makes CNNs an essential enabler of intelligent automation. Despite these achievements, the widespread deployment of CNNs has also highlighted two major challenges: **efficiency** and **security**.

On the efficiency side, CNNs are notoriously resource-hungry. Training SotA models involves billions of operations and extensive computing infrastructures, while inference still demands substantial computational power and memory [12]. This creates issued for deploying CNNs in edge computing, mobile platforms, and Internet of Things (IoT) devices, where resources are scarce, latency constraints are strict, and power consumption is a primary concern. To address this issue, several methodologies have been proposed to improve efficiency [13], including pruning [14], low-rank factorization [15], weight quantization [16], and the Approximate Computing (AxC) design paradigm [17]. The AxC deliberately replaces exact arithmetic components with approximate ones, trading accuracy for reductions in hardware overhead, and exploiting the error resilience of neural networks.

On the security side, CNNs have been shown to be vulnerable to adversarial attacks. Szegedy *et al.* [18] revealed that adding carefully crafted but imperceptible perturbations to images could drastically change the output of a neural network. Since then, a wide range of adversarial attacks have been proposed. Gradient-based methods, such as Fast Gradient Sign Method (FGSM) [19], Basic Iterative Method (BIM) [20, 21] and Projected Gradient Descent (PGD) [22], exploit model gradients to generate perturbations efficiently. Optimization-based attacks, including DeepFool [23] and the Carlini-Wagner (CW) method [24], compute minimal perturbations that cross the decision boundary, but at the cost of high computational effort. Finally, query-based black-box strategies such as the One-Pixel (OP) [25] and Square Attack (SQ) [26] attacks rely only on model outputs to search for adversarial perturbations.

Despite their success, these attacks present practical limitations. Both gradient-based and query-based methods often generate adversarial images that, while are as close as possible to the original ones, can still produce visible perturbations to human observers. Optimization-based methods, although more precise, are computationally expensive and less practical in real-time scenarios. Black-box methods are more realistic, but typically require a very high number of queries, making them inefficient and detectable in practice. These limitations highlight the need for stealthier, more efficient, and more transferable adversarial strategies, fueling ongoing research in adversarial machine learning.

Furthermore, efficiency and adversarial robustness have been studied in isolation. Efficiency techniques are usually evaluated in terms of accuracy and resource savings, while adversarial robustness is tested on full-precision floating-point networks. Yet, these two aspects are deeply intertwined. Efficiency-oriented methodologies alter the numerical behavior of neural networks: quantization discretizes weights and activations, introducing step-like decision boundaries, while AxC deliberately injects hardware-level

noise into the computation. Although these changes often preserve accuracy on clean data, they may profoundly affect adversarial robustness, either enhancing resilience to certain attacks, creating new vulnerabilities, or reducing the effectiveness of defenses. This interplay is particularly critical as AI systems move closer to deployment in resource-constrained and safety-critical settings. AxC offers promising improvements in efficiency, enabling the adoption of CNNs on platforms where power, and hardware budgets, are tight. At the same time, these systems are directly exposed to adversaries who may exploit adversarial perturbations. A deeper understanding of how approximation interacts with adversarial machine learning is therefore essential, not only from an academic perspective but also for the safe and trustworthy adoption of efficient AI systems.

1.1 Problem Statement and Contributions

Despite the impressive achievements of CNNs and their integration into safety-critical systems, two fundamental issues remain unresolved:

- **Efficiency limitations**. SotA CNNs are computationally and energy intensive, which hinders their deployment on edge and resource-constrained devices. Techniques such as quantization and AxC have been proposed to alleviate these costs, but their implications on the overall trustworthiness of neural networks are still not fully understood.
- **Adversarial vulnerability**. CNNs are inherently vulnerable to adversarial perturbations: small, carefully crafted modifications to the input that can cause misclassification with high confidence. This vulnerability raises serious security concerns in domains where incorrect predictions may have severe consequences.

While efficiency and adversarial robustness have been extensively studied in isolation, their interplay is rarely addressed. Approximation and quan-

tization alter the numerical behavior of CNNs, potentially disrupting the smoothness of the loss landscape and reshaping decision boundaries. These modifications may in turn influence how adversarial perturbations are generated, how they transfer across models, and how effective defenses remain in efficient implementations. Understanding this interplay is the central research problem of this thesis.

1.1.1 Contributions of the Thesis

This work provides a both comprehensive investigation into the relationship between AxCs and adversarial machine learning, and new algorithms that improve the stealthiness of adversarial attacks. The contributions can be summarized as follows:

1. **Novel adversarial attack strategies**. This thesis proposes new adversarial attack methodologies aimed at improving stealthiness and efficiency, by leveraging different level of knowledge of the target CNN. These attacks address limitations of existing methods, such as visible perturbations, excessive query complexity, or lack of realism, and provide further insights into the arms race between adversaries and defenders.

2. **Adversarial robustness of approximate CNNs**. This dissertation analyzes the vulnerability of CNNs approximate through the AxC design paradigm, comparing them with their floating-point original counterparts. Results show that approximate CNNs remain vulnerable to adversarial attacks, but exhibit increased robustness against them, owing to the distortion introduced by approximation.

3. **Transferability of adversarial examples through AxC**. The thesis proposes and investigates a methodology in which adversarial examples are generated on approximate networks and then transferred to other

models, including floating-point, quantized, and approximate CNNs. The results show that the AxC can unintentionally act as an enabler for highly transferable attacks.

4. **Limitations of input transformation defenses**. Finally, the last contribution evaluates adversarial detection strategies based on input transformations (e.g., rotation, scaling, blurring, contrast adjustments). While effective for floating-point models, these defenses prove largely ineffective in quantized and approximate networks, since both quantization and approximation-induced noises mask the modifications introduced by the digital transformations.

Together, these contributions provide both new imperceptible method to craft adversarial images, and the first systematic analysis of how AxC interacts with adversarial machine learning, highlighting both its potential to enhance robustness in certain scenarios and its risks as a new vector for adversarial threats.

1.2 Structure of the Thesis

The remainder of this dissertation is organized as follows:

- Chapter 2 reviews the state of the art. It first introduces basic concepts related to AI, Machine Learning (ML), and CNNs, along with formal definitions. Subsequently, it reviews adversarial machine learning, covering the most several attack and defense methodologies proposed in the literature, and then presents the challenges addressed trough this thesis.

- Chapter 3 discusses novel adversarial attack strategies developed during this research. These contributions include new algorithms designed to improve the stealthiness and efficiency of adversarial examples,

addressing some of the limitations of existing attack methodologies, in both the gray and the black-box scenarios.

- Chapter 4 investigates the interplay between the AxC and adversarial robustness. The chapter is divided into three main parts: (i) an analysis of the vulnerabilities of approximate CNNs to adversarial attacks, (ii) a study of the transferability of adversarial examples crafted on approximate networks, and (iii) an evaluation of the limitations of input transformation defenses in efficient networks.

- Chapter 5 concludes the thesis. It summarizes the main findings, discusses their implications for the design of efficient yet secure deep learning models, and outlines promising directions for future research.

2 Related Work

Nowadays, Machine Learning (ML) models are deployed across diverse non-critical and critical domains, including computer vision [3], healthcare [27] and autonomous driving [8], demonstrating outstanding performance in such fields. However, widespread deployment has elevated concerns regarding their security, robustness and reliability. Indeed, such ML models remain vulnerable to a wide range of threats, aiming to mine their functionalities [7], steal their properties [28], or exploit security flaws in generated outputs [29].

Therefore, to discuss the contributions of this doctoral thesis, this chapter provides the related State-of-the-Art (SotA), across with the necessary theoretical and technical concepts. First, Section 2.1 contains fundamental concepts concerning Artificial Intelligence (AI) and ML that facilitate the reader's comprehension of the work presented in this thesis. Section 2.2 provides the reader with the concept of adversarial threat and a detailed categorization of different types of attacks based on the knowledge and capability of the adversary, along with the techniques to defend against adversarial attacks.

Each of these topics provides essential background to understand the motivations, methodologies, and significance of the original contributions presented in the following chapters.

2.1 Basic Definitions and Machine Learning Concepts

The field of computer science known as AI is concerned with the design of systems that are capable of simulating the intelligent behavior of humans. The purpose of these systems is to facilitate the execution of tasks such as perception, reasoning, decision-making, and other cognitive processes, with the objective of accomplishing these tasks in a more efficient and expeditious manner [30]. ML is a subfield of AI that focuses on the development of algorithms that facilitate the enhancement of the performance of AI systems on specific tasks through experience, typically through the process of learning from data. The scientific literature identified three different learning paradigm within ML: (i) in the context of **supervised learning**, the model is trained on labeled input-output pairs, in order to generalize the relationship between inputs and their respective outputs. The objective of the supervised learning is to classify new data that the model has not previously encountered during the training phase, and assign it the correct labels. (ii) **Unsupervised learning** involves identifying hidden structures and patterns in unlabeled data, with the goal of automatically assigning labels to the input data. Finally, (iii) **reinforcement learning** is concerned with learning optimal actions to perform through trial-and-error interactions with an environment to maximize cumulative rewards [31]. Among all MLs models, Artificial Neural Networks (ANNs), ore more simply Neural Networks (NNs), are the most widespread model. NNs are computational architecture inspired by the biological neural systems found in the human brain. Indeed, they are composed of nodes, i.e. neurons, that are interconnected and change information between each other, usually organized in layers [32]. Figure 2.1 depicts the basic model a neuron. A generic neuron performs the weighted sum between multiple inputs x_i and weights w_i, $i \in \{0, \dots n\}$. In the event that the weighted sum exceeds a designated threshold Υ, the neurons undergo activation, subsequently transmitting

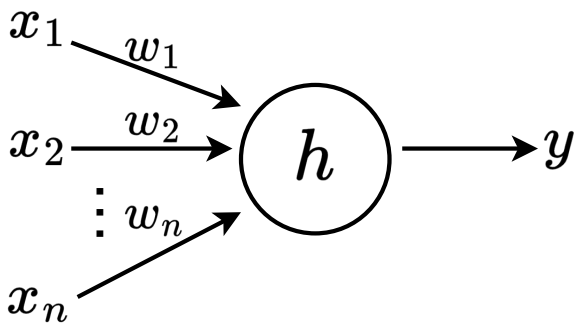


Figure 2.1: The basic model of a neuron within a neural network.

the sum trough an activation function $h(\cdot)$, in order to produce the output y, i.e., the activation. Equation (2.1) depicts the mathematical model of a neuron.

$$y = h\left(\sum_{i=0}^{n} x_i \cdot w_i - \Upsilon\right) \tag{2.1}$$

Since neurons, and more in general NNs perform complex tasks that are mostly non linearly separable, the most common activations functions are the non-linear ones, such as Rectified Linear Unit (ReLU), Sigmoid and so forth [33]. Finally, in the training phase, a designed function is used to adjust the weights w_i with back-propagation algorithm, typically the Stochastic Gradient Descent (SGD) [34] or its variants.

2.1.1 Deep Learning and Deep Neural Networks

Deep Learning (DL) is a specialized subfield of ML that allows learning from row data by automatically discovering the most suitable internal representation, of such data, needed for the final object, either the detection or the classification. In specific, DL models, known as Deep Neural Network (DNN), are NNs that are composed of at lest three layers: one input-layer, multiple hidden layers, and an output layer. Each layer of a DNN transforms the

representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. This characteristic is called *feature learning*. Complex functions, such as image classification, natural language processing ans so forth, can be learned by composing enough such transformations [1, 2].

Convolutional Neural Networks (CNNs), a particular class of DNNs, particularly effective for processing data with a grid-like topology, such as images. Indeed, they are employed in a wide range of research fields, including image processing [35], biomedical applications [6], biometrics [9] and many others. The core of such networks are the convolutional layers, composed of neurons that leverage the concept of convolution between the input and a filters – or kernels – to perform the feature learning. The filters used for the convolution are learned during the training phase, together with the other weights. When several convolutional layers are stacked in sequence, the whole network starts learning how to extract a hierarchical set of features, from a low to a high level of details. Figure 2.2 from [7] provides an example of a CNN. Typically, the structure of CNNs is divided in two separate sections. (i) The **Feature Learning Section** extracts feature within the input image, and learn the most important ones. The operation of learn features is mostly performed by convolutional layers. (ii) The **Classification Section** provide the final classification of the input image, by classifying the features extracted by the Feature Learning Section.

Formal Definitions

For this thesis, the following mathematical notation is adopted while referring to DNNs, and in particular to CNNs. Let $\mathcal{F} : A \rightarrow B$ be the function that map an input space $A \subseteq \mathbb{R}^n$ – whose data distribution is $\mathcal{D}$ – to an output space B. A CNN aims to approximate the true function $\mathcal{F}$ with an inference function f, that generalizes well to new, unseen data. Such a hypotheses function is learned by training the CNN on a labeled training set $\mathbf{T} = \{(t_t)\}_{t=1}^{T} \subseteq A$,

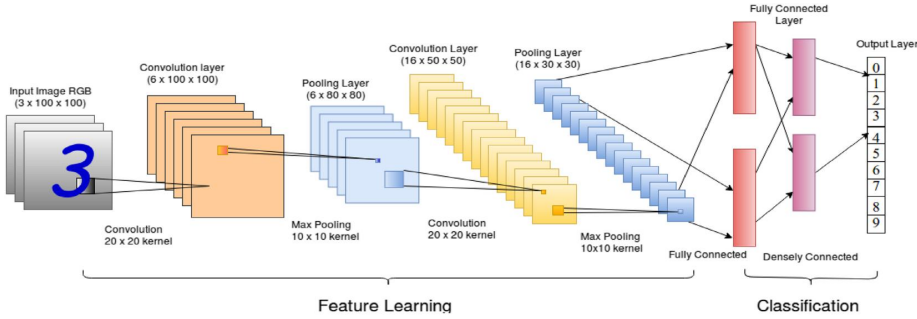


Figure 2.2: An example of CNN trained to recognize handwritten digits from the MNIST dataset [36]. Please refer to [7] for further details related to the Figure.

whose labels are $\mathbf{Z} = \{(z_t)\}_{t=1}^{T} \subseteq B$. In a classification problem, the generic label z_t is a class belonging to a set of M classes $\mathbf{Y} = \{(y_j)\}_{j=1}^{M}$. Subsequently, the better the CNN is trained, the better inference function f generalizes on new data and approximate the true function $\mathcal{F}$. In specific, the CNN adjusts all the training parameters, referred as θ – such as as the weights – during the training process with a cost or loss function $\mathcal{L}$. Without delving excessively into the particulars, during the training the CNN computes the difference (the loss) between the true label, of an input sample, and the one predicted by the network itself. Consequently, the parameters are adjusted in order to reduce the differences between the true and predicted labels.

To demonstrate the efficacy of the function f, once trained, the CNN is tested on a test set $\mathbf{X} = \{(x_i)_{i=1}^{N}\} \subseteq A$. The goal of the network is to infer the correct class y_j for each sample within the test set. Subsequently, the CNN performs the operation described in Equation (2.2), for a generic test sample x.

$$\rho = f(x) \tag{2.2}$$

$\rho = \{(\rho_j)_{j=1}^{M}\}$ is the output vector returned by the inference function f, and, in specific, by the last layer of the the CNN, that typically perform the *Softmax* function [37]. As a consequence, each entry $\rho_j \in \rho$ is in the range $[0,1]$, and the entries add up to 1, i.e.,

$$\sum_{j=1}^{M} \rho_j = 1 : 0 \leq \rho_j \leq 1 \; \forall j \tag{2.3}$$

Therefore, the output can represent the categorical probability distribution of the M classes, hence the generic entry ρ_j represents the probability that the test sample x belongs to the corresponding class y_j. In addition, such entries ρ_j are known as *soft labels*, since they are generated from the softmax layer – note that the scientific literature refers to the outputs of the layer before the softmax one as *logits* – . Assume that y is the correct (labeled) class for the sample x. The class y_a – namely, *hard label* – inferred by the CNN trough the function f, for the sample x is computed as in Equation (2.4):

$$y_a = \arg\max(\rho) = \arg\max(f(x)) \tag{2.4}$$

It follows that the likelihood of x belonging to y_a is $p_a = \max(\rho)$. In the case that the inferred class corresponds to the correct one, i.e., $y_a \equiv y$ as in Equation (2.5), we can sat that the network correctly classifies x.

$$y_a = \arg\max(f(x)) = y \tag{2.5}$$

The more test samples $x_i \in X$ the model correctly classifies, the better the function f has approximated the true function $\mathcal{F} : A \rightarrow B$ during the training. Finally, note that the the above notations are valid for both the test samples x and the training samples t. Indeed, the output vector ρ is generated as in Equation (2.2) for also the training samples, subject to the condition of Equation (2.3). In addition, a class y_a is inferred also for each train sample

during the training process, as in Equation (2.4). A list of symbols with the corresponding definitions, used in the whole thesis, is given in Table 2.1 [1].

Table 2.1: List of symbols and the corresponding definitions related to DNNs and CNNs.

Symbol	Definition
A	Input domain
$\mathcal{D}$	Distribution of data within the input domain A
B	Output domain
$\mathbf{T}$	Training Set
t	Generic train sample
T	Number of samples in the training set
f	Inference function
$\mathbf{Y}$	Classes (labels)
M	Number of classes
$\mathcal{L}$	Loss function
θ	Training parameters
$\mathbf{X}$	Testing set
x	Generic test sample
N	Number of sample in the test set
ρ	Output vector of the inference function f
ρ_j	Prediction score associated to the generic class y_j
y_a	Inferred class for a a train sample t or test sample x
ρ_a	Prediction score associated to the inferred class y_a
y	Correct class of a train sample t or a test sample x
ρ^*	Prediction score associated to the correct class y

2.1.2 Generative Artificial Intelligence

While traditional ML/DL models are primarily designed for classification and predictive tasks, an increasingly important trend in recent years is the development of models capable of generating new data sample. This emerging class of models, referred to **Generative AI**, leverages the advancement in ML and DL to create novel content [39], such as images [40], text [41], audio [42] or code [43], that mimics the patterns and the characteristics of the

[1] A detailed explanation of DNNs and CNNs is beyond the scope of this work. For a more complete and detailed explanation please refer to [1, 38].

training data. Generative AI can employ different techniques like Variational Autoencoder (VAE) [44], Generative Adversarial Network (GAN) [45] and others to achieve the desired generation capabilities. Among these generative AI models, Large Language Models (LLMs) are specifically designed to generate human-like language in response to a given context or prompt, namely, *in-context learning* [46]. These models are trained on massive amounts of textual data, by leveraging unsupervised training techniques to learn the statistical patterns of human language [47, 48]. Recently, LLMs are largely employed to generate source-code, such as OpenAI Generative Pre-Trained Transformers (GPTs) series[2] [49], Github Copilot[3] and Google Bidirectional Encoder Representations from Transformers (BERT) [50], achieving outstanding performance this task. In this thesis, LLMs are investigated particularly in the context of output security, focusing on mitigating vulnerabilities in AI-generated code [4].

2.2 Adversarial Threats and Defense in Machine Learning Systems

ML models, particularly DNNs, have achieved outstanding performance across a wide range of domains. However, security and integrity of the applications that adopt DNNs pose great concern. In essence, DNNs are vulnerable to adversarial attacks, a class of attacks introduced by Szegedy *et al.* in 2013 [18]. Such attacks involve the meticulous creation of perturbed, yet legit, inputs that are visually similar to the original inputs but can force a trained model to produce incorrect outputs [7, 51–53].

[2] https://chatgpt.com/

[3] https://github.com/copilot/

[4] A detailed explanation of Generative AI and LLMs is beyond the scope of this work. For a more complete and detailed explanation please refer to [39, 46].

Figure 2.3 provides a visual example of an adversarial example in the field of computer vision, and in particular in an image-classification task. In the example, author Goodfellow *et al.* [19] targeted GoogleLeNet [54], trained to classify images from the ImageNet dataset [55], with the Fast Gradient Sign Method (FGSM) attack they proposed in the paper. Please, note that details regarded the mentioned attack will given in the following. In specific, GoogleLeNet classified the panda image with the corresponding – and correct – class, although with a low confidence-score. Next, the authors of the paper added a perturbation to the panda image, resulting in an adversarial image that is equal to the original one, for human eye. However, GoogleLeNet produced an incorrect output while classifying the adversarial image. Indeed, it classified such an image with the *gibbon* class, with a confidence-score that is greater than 90%.

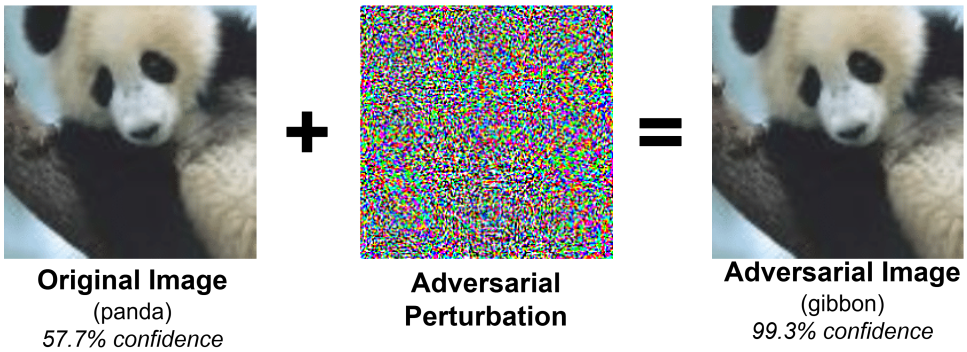


Figure 2.3: Example of an adversarial image from the Reference [19]. The authors apply a perturbation to a panda image from the ImageNet dataset [55], resulting in an adversarial image that is visually indistinguishable from the original one. However, although the target DNN GoogleLeNet [54] correctly classifies the original image, it produces an incorrect output-label while classifying the adversarial image.

2.2.1 Symbols and Formal Definitions

In this thesis, we mathematically refer to a perturbed example by resorting to Equation (2.6). The original/clean sample is denoted by x, δ_x represents the adversarial perturbation, while $\hat{x}$ denotes the perturbed sample, that is, a perturbed version of the clean sample.

$$\hat{x} = x + \delta_x \tag{2.6}$$

The perturbed sample $\hat{x}$ is considered an adversarial sample whether results in a misclassification by the target model. Assume that the sample x is correctly classified by the model, as in Equation 2.5, and y is its true class. We denote as $\hat{\rho}$ the output vector of the model while classifying the perturbed sample $\hat{x}$ as in Equation (2.2), and as $\hat{y}_a$ subsequent inferred class, as in Equation (2.4), $\hat{x}$ is an adversarial sample if the condition of Equation 2.7 is met. As a consequence, we denote as x^{adv} the adversarial sample.

$$\hat{x} \equiv x^{adv} \iff \hat{y}_a \neq y \tag{2.7}$$

In specific, an adversarial sample is a perturbed version of the clean sample x, in which the perturbation δ_x is large enough to result in a misclassification. In such a scenario, $\hat{x}$ crosses the decision boundary, resulting in its classification as $\hat{y}_a \neq y$. Furthermore, we will denote as $\hat{\rho}_a$ and ρ_a^{adv} the likelihood that the perturbed sample $\hat{x}$ and the adversarial one x^{adv} belongs to the predicted classes $\hat{y}_a$ and y_a^{adv}, receptively, computed as in Equation (2.2). Finally, $\hat{\rho}^*$ and $\rho^{adv,*}$ refers to the likelihood that $\hat{x}$ and x^{adv}, respectively, belong to the true class y associated to the original image x Table 2.2 depicts symbols related to adversarial samples.

Note that throughout this thesis, the expressions *adversarial sample*, *adversarial example*, and *adversarial image* are used interchangeably. They all refer to inputs deliberately designed to cause incorrect predictions in machine learning models.

Table 2.2: List of symbols and the corresponding definitions related to adversarial samples.

Symbol	Definition
$\hat{\mathbf{x}}$	Perturbed sample
δ_x	Perturbation
$\hat{\rho}$	Output vector while classifying $\hat{x}$
$\hat{y}_a$	Class inferred while classifying $\hat{x}$
$\hat{\rho}_a$	Prediction score associated to the inferred class $\hat{y}_a$
$\hat{\rho}^*$	Prediction score associated to the correct – true – class y
x^{adv}	Adversarial sample
y_a^{adv}	Class inferred while classifying x^{adv}
ρ^{adv}	Output vector while classifying x^{adv}
ρ_a^{adv}	Prediction score associated to the inferred class y_a^{adv}
$\rho^{adv,*}$	Prediction score associated to the correct – true – class y

2.2.2 Adversarial Threat Model

In order to evaluate the security of ML models, it is mandatory to consider the **goal** of an attacker, his **capabilities**, and the **knowledge** he has regarding the target systems. It is possible to taxonomize adversarial attacks based on these three different concepts. However, before delving in the taxonomy of adversarial attacks, in the following we provide an illustrative example of a real-life ML system.

In general, a system that utilizes ML models can be regarded as a data processing pipeline. For illustrative purposes, one may consider a self-driving car that is required to come to a halt at a *stop sign*. A vehicle of this type is equipped with a sensor (a camera, in this instance) that captures an image of the stop signal. Subsequently, the image is processed by a ML model, such as a CNN, to facilitate its classification. In the event that the CNN correctly classifies the image as a stop signal, the output of the CNN will enable the generation of the *stop command*. Consequently, this command will be transmitted to an actuator, thereby initiating the cessation of vehicle operation. This system can be seen as a data-processing pipeline. Indeed, (i) the first operation is to collect the data (stop signal collected by the camera). Subsequently, (ii) such data are transferred to the digital domain – an RGB

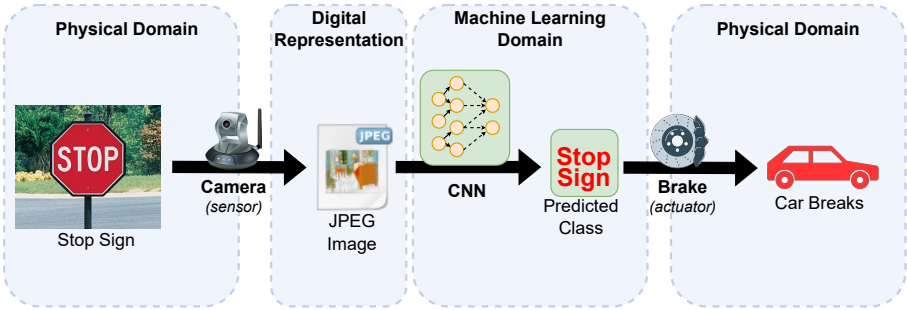


Figure 2.4: A generic and simple self-driving car example to illustrate a ML system as a data processing pipeline.

image of the stop signal is generated – and, consequently, (iii) processed by the ML model to produce an output. Finally, (iv) based on the previous action, an action is taken. Such a real-life application is widespread in the scientific literature [7, 56, 57] either for research reasons, or to provide the reader with an illustrative example, as in this thesis. Figure 2.4 depicts an illustrative representation of the above-mentioned real-life car application. By resorting to the above illustrative example, and considering a ML model as a data-processing pipeline, an adversary can attack the system by either manipulating the collection and processing of data, corrupting the ML model itself, or alter the output.

In the following, a taxonomy of adversarial attacks based on *capabilities*, *knowledge*, and *goal* of an attacker is provided.

Adversarial Capabilities

The capabilities of an attacker refer to is strength, that depends on the level of access that the attacker has to the system and its data [56].

In the *training phase*, the adversary has access to either the training dataset of training algorithm of the ML model, in order to learn, influence or corrupt the model itself.

The weakest attack is simply the accessing a summary, a partial of all of the training data. Indeed, depending on the quality and volume of the training data, an attacker can build and train a substitute model that behaves as similar as possible to the original one, in order to transfer the learning. Furthermore, adversarial attacks during the train phase could alter the target model and corrupt its functionalities. Indeed, SotA denotes such kinds of attacks as **poisoning attacks** [58–60]. They mostly modify the training data by either injecting manipulated inputs into the training dataset [61], or directly modifying training samples. In addition, an adversary can modifying training samples by either manipulate their labels [62], or directly the features of the training samples itself. Hence, poisoning attacks mine the integrity of the target model.

In the *testing phase*, attackers target the ML model during the inference. There are two kinds of testing-phase attacks. **Evasion attacks** evade the ML model by forcing it in producing incorrect outputs. In particular, attackers carefully craft malicious input sample in order to cause the target model misclassifies the sample [63]. **Exploratory attacks**, on the other side, assume no knowledge about the target mode. Instead, an attacker queries the model in order to gain as much knowledge as possible about the learning algorithm of the ML system and patterns in the training data.

Adversarial Knowledge

It is possible to classify adversarial attacks basing on the knowledge that the adversary has about the target model.

White-box attacks assume a complete knowledge of the target system. Indeed, the adversary has access to both the training and test datasets, and details about the models, including the architecture, the parameters and learning algorithm, as well as the activation and loss functions [64, 59].

Gray-box attacks assume limited knowledge about the model, by having a partial access to its architecture or defense strategy [65]. However, gray-box

approaches remain under-explored by the scientific literature for the high computational costs. Furthermore, the notion of gray-box knowledge is often misinterpreted. Indeed, in reality, such attacks require access to sensitive information pertaining to the ML system, which is typically characterized by white-box knowledge [66–68].

Finally, **black-box attacks** assume no knowledge about the target model. In such a setting, the adversary has access to only either inputs or outputs of the model, or to both of them. Hence, the attacker queries the model as an oracle, in order to gain information on it and craft the adversarial samples [69, 52]. Black-box attacks are further divided into the following categories [7]: (i) *Non-adaptive*: the attacker has access to the distribution of the training-data of a target model f. In these settings, the objective is to train a surrogate model f' with samples from the training-data distribution to approximate the target model. Subsequently, the attacker craft adversarial samples by targeting the surrogate model with white-box attacks, and applies these crafted inputs to the target model in order to cause a misclassification. (ii) *Adaptive* black-box attacks access the target model as an oracle, without knowing details related to the training procedure. In specific, the adversary queries the target model as an oracle with arbitrarily-chosen input samples in order to obtain the related labels as outputs. Subsequently, the adversary trains a surrogate model with the the couple (arbitrarily-chosen) inputs-labels. Finally, as for non-adaptive attacks, the adversary craft adversarial samples by white-box attacking the surrogate model, and use such samples to force a misclassification in the target model. (iii) *Strict* attacks: this is the case when an attacker can collect the input-output pairs directly from the target model, but it has not the possibility to alter the inputs and observe the changes in outputs.

Note that both *non-adaptive* and *adaptive* are **transfer-based** adversarial attacks. Typically, ML systems are trained with the same algorithm, such as SGD or its variants, use the same activation functions, such as ReLU or

its variant, and leverage similar data-augmentation techniques to increase the size of training data. As a consequence, such systems share the same vulnerabilities, meaning that every ML model is vulnerable to adversarial attacks. Therefore, different models with different architecture and trained with different datasets can misclassify the same adversarial samples. In this case, an adversary can target a source (or surrogate) model with an adversarial algorithm (with white-box attacks, for instance), and then *transfer* crafted samples to the target model in a black-box settings [70]. Such a property is known as *transferability* [71]

Adversarial Goal

Finally, adversarial attacks can be classified based on the goal of an attacker. The objective of an attacker is inferred from the impact that an adversarial sample has on the integrity of the output[72, 7, 52].

An adversary could **reduce the prediction-confidence** of the target model. Indeed, by resorting to the example of Figure 2.4, the target ML model can correctly predict a *stop* sign with a lower confidence, hence with a lower probability that such a sign belongs to the *stop* class.

In addition, the goal of an attacker can be an **untargeted** misclassification: the adversary tries to alter the output classification of an input sample to some other class. In the example of Figure 2.4, a legitimate *stop* sign will be predicted as any other class different from the true one.

Conversely, **targeted** attacks aims to craft adversarial inputs that are classified by the model into a particular class. In the example of Figure 2.4, any input sign could be modified in such a way that the model classifies it as a *go* sign.

Finally, in the **source-target** misclassification, the attacker modifies a particular input sample in such a way that the model misclassifies it with a predefined target class. In the example of Figure 2.4, a *stop* sign could be modified in such a way that the model classifies it as a *go* sign.

2.3 Evasion Attacks on CNNs

One of the core contributions of this thesis are evasion adversarial attacks in the computer vision fields, and, in particular, in image classification tasks. Therefore, in the following, we provide a review of evasion attacks from the scientific literature, i.e., test-phase attacks aiming in causing a misclassification, in both the white and black box settings. Finally, such a Section will focus exclusively on adversarial attacks that target CNNs. Before analyzing SotA attacks, please note that the objective of such attacks are both (i) to find an adversarial image that fools the target network, and (ii) and to minimize the distance between the original image and the adversarial one, in order that the latter is as similar as possible to the former. The distance between the two image is generally computed with the L_p norm, as in Equation (2.3).

$$||x - x^{adv}||_p = \left(\sum_{i=1}^{n} |x_i - x_i^{adv}|^p \right)^{\frac{1}{p}}$$

In specific, the more widespread ones are:

- the L_0 distance measure the number of coordinates i that differs between the original image and the adversarial one. For images, the L_0 norm measures the number of pixels that differ between the two images.

- The L_1 norm measures the total variation between x and x^{adv}, for all the components (pixels). Although, such a distance metric is less widespread than the others [53].

- The L_2 norm measures the standard Euclidean distance between x and x^{adv}. In case of images, such a distance can remain small when there are many small changes to many pixels.

- The L_∞ distance measures the maximum change to any of the coordinates (pixels). For images there is a maximum budget, since a pixels

has a maximum and a minimum value – typically in the range $[0, 255]$ for RGB images –. This means that each pixel is allowed to be altered up to this maximum budget, with no limit on the number of pixels that are modified.

2.3.1 White-Box Attacks

Adversarial attacks in DL were introduced by Szegedy *et al.* in [18]. They found that adding a small perturbation δ_x to an input image x would result in an adversarial image x^{adv}. In specific, authors wanted to find the minimum perturbation that allows for the adversarial image to being classified as a target class y^*, by solving the following optimization problem:

$$\min ||\delta_x||_2, \; s.t. \; f(x+\delta_x) = y^*, \; x+\delta_x \in A$$

Please, refer to Tables 2.1 and 2.2 for symbols and definitions. However, such a problem is a hard one, hence authors approximated it by using a box-constrained Limited-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm (L-BFGS)[73]. Indeed, the L-BFGS yeld the exact solution in the case of convex problems. However since NNs are non-convex in general, their solution is an approximate one. The authors observed that L-BFGS allow for generating adversarial image with imperceptible adversarial perturbation, indeed the adversarial image is almost identical to the original one. Also, they observed that their adversarial images transfer to other models. However, although their method has good performances, it is computationally intensive since authors leveraged a linear search method. Also, L-BFGS assumes a cross-entropy loss function.

To further improve the performance of L-BFGS, Carlini and Wagner [24] proposed the Carlini-Wagner (CW) attacks, that are optimization-based attacks. Indeed, while L-BFGS relies on the cross-entropy loss, CW involves a margin loss as the loss function, that can be customized by the attacker.

Indeed, their attack supports several loss functions. In addition, authors imposed a box constraint on the perturbation δ_x to ensure that the perturbed image $\hat{x}$ is still a valid image, and explored different methods that allow to use also optimization algorithms that do not natively support box constraints. However, in order to not be limited by the box-constraint, but at the same time crafting valid perturbed images, authors operated a change of variable via the `tanh` transformation of the perturbed image. Finally, authors designed the attacks for the L_0, L_2 or L_∞ norm as distance metrics.

Inspired by the CW attacks, Chen *et al.* [74] proposed Elastic-Net Attack against DNNs (EAD), that is, a targeted box-constrained attack that leverage the elastic-net regularization [75] – a regularizer that linearly combines L_1 and L_2 penalty functions – to craft adversarial examples. Authors noted that using the L_1 penalty for the perturbation improves the transferability of the crafted adversarial images. However, the `tanh` transformation leveraged by the CW attacks [24] craft adversarial examples that are insensitive to some changes of the L_1 penalty, that is also not-differentiable. Therefore, authors of [74] adopted the iterative shrinkage-thresholding algorithm as optimization algorithm.

The following attacks are known as gradient-based attacks, since they leverage the gradient of the loss function – a fisrt-order information – to craft adversarial images.

Moosavi-Dezfooli *et al.* [23] proposed the DeepFool attack, that is, an untargeted attack technique optimized for the L_2 distance metric. Since a multiclass classifier can be viewed as an aggregation of binary classifiers, authors first designed the attach for a binary classification problem. They also assumed the differentiability of the loss-function. In such a case, at each iteration of the attack, the function f is linear in a neighborhood of the modified image $\hat{x}_i$, and classes are lineally separated by an hyperplane. Subsequently, authors extended the formulation for multiclass classifiers by iterating the procedure of the binary classification case. Finally, they

extend the linear multiclass case to a general non-linear multiclass problem, by operating in a greedy way. However, as a consequence of using greedy algorithm, it is not guaranteed that DeepFool converges to the optimal solution. Note that, although DeepFool is designed and optimized to use the L_2 norm as a distance metric, authors extended the attack for any L_p norm.

Moosavi-Dezfooli *et al.* discovered the possibility of creating adversarial images with universal perturbations, i.e., perturbations that are not dependent on the target image [76]. In specific, authors propose an iterative algorithm for the purpose of computing a perturbation that results a small set $\mathcal{I}$ of samples within the training dataset to be misclassified. As a consequence, such a perturbation is able to generate adversarial images with the whole training dataset, although the $\mathcal{I}$ is much smaller than the training dataset.

Goodfellow *et al.* [19] proposed FGSM, that is, an efficient white-box algorithm to craft adversarial samples with a single step. In specific, given the Equation (2.6), the perturbation is computed as in Equation (2.8).

$$\varepsilon \cdot sign(\nabla_x \mathcal{L}(\theta, x, y)) \tag{2.8}$$

In specific, $\mathcal{L}(\theta, x, y)$ is the value of the loss function related to the input sample x and its true class y, linearized around the current value of the parameters θ. $\nabla_x \mathcal{L}(\cdot)$ is the gradient of the loss function, while $sign(\cdot)$ is the sign of the gradient. Finally, ε is a small error that is used to constraint the final perturbation. Hence, FGSM leverages the loss-function to search for the direction that increases the classification error. In other words, FGSM generates an adversarial image by adding or subtracting – depending on the sign of the gradient of loss function – the error ε to each pixel of the image x. Please note that FGSM applies with both untargeted and targeted goals. Indeed, in the targeted scenario, the idea is to maximize the probability of a specific target class which is unlikely to be the true class for a given image [20]. Although FGSM is efficient, it has a low success rate since it is

single-shot, and it would require a large (yet visible) perturbation to fool the target CNN.

Several version of FGSM were proposed in the scientific literature. Kurakin *et al.* [20, 21] proposed an iterative version of it, named Basic Iterative Method (BIM). BIM basically applies FGSM multiple times with a small step size, and clips pixel values of intermediate results after each step, to ensure that they are in a ε-neighborhood of the original image. Equation (2.9) depicts how BIM computes an adversarial image.

$$\hat{x}_{i+1} = Clip_{x,\varepsilon}(\hat{x}_i + \alpha \cdot sign(\nabla_x \mathcal{L}(\theta, \hat{x}_i, y)) \tag{2.9}$$

Here, $\hat{x}_i$ is the intermediate perturbed image during the i^{th} iteration, and α is the size of the perturbation to add to each pixel. In addition, as for FGSM, authors proposed a targeted version of BIM.

The scientific literature improved BIM with two new attacks, that are Momentum Iterative FGSM (MI-FGSM) [77] and Projected Gradient Descent (PGD) [22]. MI-FGSM integrates the momentum method [78] into FGSM. Indeed, FGSM assumes the linearity of the target decision boundary around the data point of interest. However, the linear assumption does not hold in case of large perturbations, making the adversarial example "underfits" the model. In contrast, BIM could cause the adversarial example to drop into poor local maxima and "overfits" the model, decreasing the transferability property. The momentum stabilizes the update directions of FGSM and BIM, and allows also escaping from poor local maxima. Finally, authors of [77] extends MI-FGSM to be effective against ensemble of models and in targeted settings. On the other side, PGD improves BIM by replacing the `Clip` function with the projection one. In particular, it starts to add a perturbation to the image x, and projects such perturbed image into the ball $x+S$, whose center is x and radius is ε, as in Equation (2.10).

$$\hat{x}_{i+1} = \prod_{x+S} (x_i + \alpha \cdot sign(\nabla_x \mathcal{L}(\theta, \hat{x}_i, y)) \tag{2.10}$$

Here, $S = \{\delta_x \in \mathbb{R}^n : ||\delta_x||_\infty \leq \varepsilon\}$, meaning that S is a set of all the feasible perturbations δ_x whose L_∞ norm is less-equal than the max perturbation ε. The goal of PGD is to maximize the loss in order to cause a misclassification in few steps, and with a minimum perturbation. Finally, such an attack is defined as the universal first-order attack. This means that if a CNN is robust against PGD, it his robust against all first-order attacks. Since PGD is computationally intensive, Doldo *et al.* [79] recently improved such an attack with an early termination criterion, based on the geometry of how PGD is implemented in practice. Authors save a hash of the perturbation, each iteration. Subsequently, in each new iteration, the perturbation is hashed and compared with the previous ones in order to not compute again the same perturbed images. In such a way, PGD operates with a not-fixed number of steps, and can terminate with few iterations – usually it requires round 10 iterations to find a successful adversarial sample –.

Papernot *et.al* [80] proposed the Jacobian-based Saliency Map Approach (JSMA) attack, that is, a targeted designed for the L_0 norm as distance metric. In specific, authors leverage the architecture and the weights of the target DNN, as well as the gradient of the loss function, to construct the adversary saliency map, that is, a Jacobian matrix. Such a matrix identifies the set of feature, within the input example x, that are relevant to the adversary goal. This means that the matrix identifies which are the input feature to perturb in order to craft an adversarial example efficiently, and that will me misclassified as the target class. However, the perturbations introduced by JSMA within the input image are greater – more visible – that that of DeepFool [23].

One of the drawbacks of white-box attacks is that they are not applicable to real world scenario. Indeed, in order to perform an attack, an attacker has to obtain all the information about the model, such as the internal parameters, the training algorithm or the dataset. However, in a more realistic scenario, an attacker has no access to the model, and he can query the model in black-box manner.

2.3.2 Black-Box Attacks

The following Section reviews black-box attacks from the scientific literature, with a focus on: (i) *transfer-based* attacks, that leverage surrogate models and the transferability property to attack a black-box target model, and (ii) *query-based* attacks, that query the target model as an oracle in order to craft adversarial images. Based on the information that the attacker has on the model, query-based attacks are further divided in *score-based* and *decision-based* attacks. The former leverage the whole output vector ρ of the target DNN. The latter rely only on the final decision – class – y predicted. Reefer to Table 2.1 for the symbols.

Transfer-based attacks Papernot *et al.* [57] first explored the possibility of targeting a DNN in a black box manner, without accessing the internal parameters of the networks. They assumed a partial knowledge about the training data and the output of the target DNN, and their idea was to leverage the transferability property. In specific, the proposed attack first queries the target DNN as an oracle, in order to collect the output labels. Subsequently, they craft a synthetic train dataset by leveraging the labels collected before, and with such a synthetic dataset they train a substitute model. Note that the synthetic dataset is relatively smaller than the original one, and it is composed of samples that are representative of the input domain. Therefore, the classification function f' of the substitute model is indeed an approximation of the target classification function f of the target model. Next, authors craft adversarial samples by targeting the substitute mode with JSMA [80], with the objective to mislead the substitute model. Finally, the adversarial samples are transferred to the target DNN. Indeed, the more accurately f' approximates f, the higher is the transferability-rate from the substitute model to the target one.

However, the transferability of adversarial samples is generally low. Indeed, Wu *et al.* [81] found that the transferability of adversarial examples

strongly depends on both the architecture of the source model, and on the smoothness of the loss function used for crafting adversarial examples.

Hence, several methods were proposed to improve the transferability. Phan *et al.* [82] proposed Content-aware Adversarial Attack Generator (CAG), a GAN-based attack that craft adversarial samples in a black-box setting, and both improves the transferability and reduces the computational effort. In specific, CAG is able to generate perturbations by focusing only in the critical section of the input image, and introduces a random dropout during the perturbation-generation process to improve the generality, as well as their transferability. For what pertains to GANs, they consist of two networks, a generator $\mathcal{G}$ a the discriminator $\mathcal{D}$, that are trained trough a process of adversarial competition. $\mathcal{G}$ generates synthetic data similar to the real data, while $\mathcal{D}$ attempts to distinguish between real and generated samples [5].

Serial-Mini-Batch-Ensemble-Attack (SMBEA) [84] considers the process of crafting adversarial samples to be the training of DNNs, and the transferability of adversarial example as the model ability to generalize. Authors ensemble pre-trained source models into several mini-batches. In addition, models into a batch have a similar decision boundary, although they have different architectures and parameters. Indeed, this allow to transfer adversarial samples intra-batch without "overfitting" the crafting-process to a specific model. Authors also provided three different intra-batch ensemble strategies, based on both the output space, and the feature space. Furthermore, to allow the transferability inter-batch, and not-overfitting into a single one, authors memorize the learned adversarial information of each batch.

Feature Importance-aware Attack (FIA) [85] improves the transferability by disrupting the important object-aware features that contribute the most to the final classification. However, different networks extract exclusive features to better solve the classification problem, resulting in model-specific feature representation. Therefore, disrupting object-aware features of specific model

[5]Please refer to the original work of Goodfellow *et al.* [83] for further details on GANs.

means also disrupting those exclusive features, resulting in adversarial sample whose transferability is poor. Therefore, the authors solve such problem by randomly transforming the input sample, and aggregate the gradient. Indeed, model-specific features are not robust to such a transformation, in contrast to object-aware ones.

Finally, several works from the scientific literature leverage image transformation to improve the transferability of adversarial images. Zhang *et al.* [86] proposed Mask Momentum Iterative Attack (MMIA). Such an iterative attack finds the optimal perturbation by perturbing only the sensitive regions of the most important section of the image. In specific, at each iteration, MMIA collects the gradients in order to perturb only the pixels related to the largest gradients, and masking the other pixels. Such a method improves the transferability of adversarial sample. Block Shuffle and Rotation (BSR) [87] craft a perturbed image by randomly dividing it into different blocks. Subsequently, each block is shuffled and rotated. The goal is to obtain different perturbed images related to the same clean one, and fed them to the source white-box model. In such a way, it is possible to obtain different attention heatmaps, that is, the region of the input image that contribute the most to the final classification. Consequently, collecting diverse heatmaps allow for collecting gradients with higher diversity, and improving the transferability. However, image-transformation methods [86–91] generally introduce a small gradient diversity or craft adversarial images that are too diverse from the original ones. Zhao *et al.* [92] proposed the Scale Enriching Method (SEM) to improve the transferability of adversarial examples, and solve problems of input-transformation methods. Indeed, SEM modifies inputs by changing its scale without randomization techniques, in order to introduce enough diversity in the gradients, reduce model-specific dependencies, and increase the transferability.

Query-based attacks Query based attacks estimate gradients by directly querying the target model in a black-box manner, requiring access only to

inputs and outputs. Hence, such attacks do not require substitute models. The first query-based attack was Zeroth-Order Optimization (ZOO), introduced by Chen *et.al* [93], in both the untargeted and targeted settings. Indeed, such an attack is referred as zeroth-order due the fact that it does not require first-order information, such has the gradient of the loss function. Instead, it requires zeroth-order information, i.e., inputs and outputs of the target model. The basic idea is to estimate the gradients once fed the target CNN with different images that are close to each other. Consequently, it is possible to apply classical optimization algorithms like gradient descent. Inspired to the CW attack [24]. the authors they optimized the following loss-function for the targeted scenario.

$$f(x^{adv},t) = \max\{\max_{i\neq t}[\log(\rho_i) - \log(\rho_t)] - k\} \tag{2.11}$$

Here, t is the targeted class, while i is any other class different from the targeted one. Finally, $k \geq 0$ is a tuning parameter for the transferability. Due to the properties of the $\log(\cdot)$ function, if $\max_{i\neq t}[\log(\rho_i) - \log(\rho_t)] \leq 0$, this means that x^{adv} attains the highest confidence score for the class t. In case of untargeted attacks, the loss-function in the Equation (2.11) remains almost unchanged, except for here the correct class has to be adopted. Subsequently, ZOO estimate the gradients of the loss function for all the coordinates – pixels –. Nonetheless, although authors proposed several strategies to reduce the number of queries, ZOO remains still computationally intensive. In addition, another problem is that ZOO is a score-based attack, meaning that it requires the whole output vector ρ. However, in a realistic scenario, the attacker has access only to the final decision.

Simple Black-Box Adversarial Attack (SimBA) [94] is a score-based attack that, given a predefined set of orthogonal search directions, picks among them a random one to move the image towards the decision boundary. Consequently, the attack either adds or subtracts a perturbation to the image to move it along that direction. To guarantee the query efficiency, the attack

ensures that no two directions cancel each other and diminish progress, or amplify each other and increase the norm of perturbation disproportionately. In such a way, authors ensure that $||\delta||_2 = \sqrt{T}\varepsilon$, where T is the number of total updates – queries to the target model –, where ε is the step size. Therefore SimBA trades-off the L_2 norm of the perturbation and the number of queries, by either decrease or decrease the step size ε.

Andriushchenko *et. al* [26] proposed the iterative Square Attack (SQ), designed for both the L_2 and L_∞ distance metrics, which exploits randomized search algorithms to perturb the input image. More precisely, at each step the attack applies almost the maximum perturbation (that lies on the either L_2- or L_∞-ball) before projecting – clipping – it onto to the image. Furthermore, SQ perturbs a small number of contiguous pixels shaped into squares. The dimension of the square is decreased at each iteration, while both the position of the square and the perturbation applied to each pixel of the square are selected randomly. Finally, the pixels into the square are altered with the same perturbation.

One-Pixel (OP) [25] generates adversarial attack by solving the optimization problem in Equation (2.12).

$$\begin{aligned} \max_{\delta_x^*} \ & \text{adv} := f(x+\delta_x) \\ & s.t. \ ||\delta_x|| < d \end{aligned} \tag{2.12}$$

In specific, the goal is to find the optimal an adversarial perturbation δ_x^* that maximizes the likelihood of the perturbed samples $x+\delta_x$ to be classified as a class `adv`. Furthermore, OP has to modify at most d elements of the original sample. Note that $d = 1$, meaning that at most one pixel will be modified by OP. This attack resorts to the Differential Evolution (DE) [95] to solve the optimization problem in Equation (2.12), since it is less subject to local minima than gradient descent, and it does not require fitness function to be

differentiable.

$$\begin{aligned} \delta' = \delta_a + H \cdot (\delta_b - \delta_c) \\ \delta_a \neq \delta_b \neq \delta_c \end{aligned} \tag{2.13}$$

During each iterations, OP generates trough DE a new set of candidate solutions δ' – perturbations –, according to the mutation operator in the Equation 2.13, where H is a scalar constant, and δ_a, δ_b, and δ_c are randomly chosen from the current population. In such a way, the perturbation is encoded as a *chromosome* of five *genes*, in evolutionary algorithms terminology. Two genes encode the coordinates of the pixel being altered, while the other three encode the RGB values. Although OP perturbs a single pixel within the input image, the attack is still successful since even a single-pixel change propagates into global activation shifts in deeper layers.

A drawback of the aforementioned score-based attacks is that they need the knowledge of the scores computed by the target CNN, that is, the outputs of the softmax layer. However, it is challenging for an attacker to gain access to such scores. Indeed, score-based attacks are referred to as semi black-box attacks. To overcome such limitation, the scientific literature explored the decision-based attacks. They are black-box attacks that leverage exclusively the final decision, i.e., the predicted class y_a for an input x (please refer to Equation (2.4) and Table 2.1 for notations and further details), of the target DNN to craft adversarial samples.

The Boundary Attack [96] is an iterative decision-based attack that craft the adversarial sample by applying a large perturbation to the input image. Subsequently, it iteratively reduces the perturbation while staying adversarial. In specific, the attack first explores the neighborhood of the target image by choosing an adversarial version of it with a relatively huge perturbation, in either the untargeted or targeted scenario. Next, the attack chose a direction from an iid Gaussian sampling-distribution and projects the adversarial image on a sphere around the original image, along the selected direction. Authors

of [96] denote as *orthogonal perturbation* the perturbation that allows for moving the image along such a direction. Then, the image is small-moved towards the original image. The above moving operations are repeated for k steps, until the L_2 distance between the original image and the adversarial one is smaller than a threshold. The goal is to move the adversarial image as near as possible to the classification-boundary of the original image.

HopSkipJump [97] is an iterative query-efficient decision-based attack designed for L_2 and L_∞ distance metrics, in both the untargeted and targeted scenarios. It generates adversarial examples by estimating the direction of the gradient with only the predicted label. The attack starts from an initial adversarial image – either found by a random search or by applying a large perturbation – and iteratively refines it by approximating the decision boundary of the target classifier. At each iteration, the attack first performs a binary search between the original image and the current adversarial one to find the closest image to the boundary. Second, it estimates the direction of the gradient by applying random perturbations and query the model to observe how the output label changes. Finally, once approximated the gradient, HopSkipJump updates the adversarial sample by moving it slightly in the direction that maintains the adversarial status while reducing either the L_2 or L_∞ distance to the original image.

2.3.3 Challenges

The preceding sections reviewed evasion adversarial attacks from the scientific literature, with particular attention being directed towards both white box and black box attacks. Nevertheless, there are as yet unresolved challenges that this thesis must confront.

Challenge 1 White-box generally have a high success rates; however, they are impractical in a real-world scenario. Indeed, they either require a fully access to the target model and it's internal parameters [18, 24, 74], or

to details regarding the training procedure [23, 19–21, 77, 22], or both of them [80].

Challenge 2 On the other side, black box attacks are more realistic, but also more computationally-intensive than white-box attacks. Indeed, they require to built and train substitute models, and query them with white-box methodologies [57, 82, 84–87, 92]. Also, they require exclusively either the output vector from the target model [93, 94, 26, 25] – although losing in applicability –, or the output label [96, 97]. A significant disadvantage of black-box attacks is that as the amount of information utilized in the execution of an attack decreases, an increased number of queries are necessary to the target model. As a result, the attacks become both more computationally intensive, and more detectable by defending systems.

Challenge 3 Most adversarial attacks methodologies, especially the query-based black-box ones, craft adversarial images by perturbing pixels within the input images that are not important to the final classification. Indeed, in most cases, modifying such pixels either (i) increases the number of queries to the model, that could be reduced by perturbing only the pixels that contribute the most to the final classification, or (ii) introduces a perturbation that could be visible in the final adversarial image.

Challenge 4 Although adversarial attacks should introduce not-visible perturbation to the input image, usually the introduced perturbation is quite visible, especially in black-box scenarios.

Challenge 5 All the attacks from the literature suffer from non reproducibility. Indeed, they mostly leverage stochastic methodologies, for tuning the perturbation to apply to the input image.

2.4 Adversarial Defense

This thesis provide insights also related to adversarial defense methodologies in the field of computer vision, especially in the image-classification task. Therefore, in the following, we provide a review of adversarial-defense mechanisms from the scientific literature. In particular, based on the defense objective, the taxonomy of adversarial defense methodologies includes [98, 7, 52, 51]:

- **Mitigation** methods, i.e., *proactive* methods that enhance the robustness of ML models against adversarial attacks;
- **Detection** methods, i.e., *reactive* methods aiming to detect adversarial attacks before they reach the target ML models.

2.4.1 Mitigation Defense-Methods

During the past years, several mitigation methodologies were introduced to mitigate the effect of adversarial samples. Such methods include the (i) modification of the training process, (ii) the transferring of the adversarial robustness, and (iii) the modification of input images.

Adversarial Training

Adversarial training techniques enhance the model robustness by leveraging a training dataset composed of both benign and adversarial examples. In specific, given the train sample t classified as y, the adversarial training leverages an adversarial attack algorithm to craft an adversarial sample t^{adv} still classified as y. Consequently, the tuple (t^{adv}, y) is inserted in the train dataset T, resulting in a new train dataset T^* composed of both benign train samples and adversarial ones. Finally, the DNN is retrained using this new train dataset. Note that the adversarial training is the most well-known (and explored) defense method against adversarial attacks [98]. Such a technique

was introduced by Goodfellow *et al.* in [19]. Indeed, in authors provided both the FGSM – discussed in the pervious section – and the defense against such an attack – they leveraged FGSM itself to perform adversarial training –.

To enhance the robustness against adversarial attacks, the adversarial training is typically addressed as a min-max optimization problem [99], first proposed in [100]. The inner maximization problem aims to find an adversarial version, with a bounded perturbation, of a given input sample that achieves a high loss. That is, the inner problem is the problem of attacking the DNN itself. Authors of [100] use two gradient-based methods to approximate the inner maximization problem, based on the output vector ρ and the probability of incorrect predictions, since it is computationally intensive to solve the original inner maximization problem. On the other hand, the outer minimization problem aims to train the model parameters to minimize the adversarial loss given by the inner attack problem. This is precisely the problem of training a robust model using adversarial training techniques. The adversarial training problem is mathematically described as in Equation (2.14).

$$\min_{\theta} \mathbb{E}_{(t,y)\sim\mathcal{D}} \left[\max_{||\delta||\leq\varepsilon} \mathcal{L}(f_\theta(t+\delta), y) \right] \tag{2.14}$$

Here, $\mathbb{E}_{\mathcal{D}}[\cdot]$ is the expected population risk, that measures how much the target model misclassifies a training sample with the parameters θ.ε is an upper bound for the perturbation. For the other symbols please refer to tables 2.1 and 2.2.

The authors of PGD [22] also proposed an adversarial training technique that make models robust to a wide range of adversarial examples. In specific, they solved the inner problem of Equation (2.14) with PGD, and the outer problem with SGD. In such a way, authors discovered that by increasing the size of the network, its robustness is increased only against one-step attacks, e.g., FGSM. In addition, leveraging FGSM to solve the inner problem causes the target DNN overfits to these adversarial examples.

One method of enhancing the effects of adversarial training is to perturb the intermediate layers in addition to the input one. The authors of [101] generate adversarial images, aimed to adversarial-train the CNN, by perturbing the activations in intermediate layers. In addition, they proposed a gradient-accumulation method: the perturbation used to perturb an image – and craft the adversarial sample – is leveraged to perturb the next sample. The latter approach allows for reducing the computational overhead of the adversarial training.

Another methodology to reduce the overhead of the adversarial training is to to update both the model parameters and the image perturbation during the backward step of the training procedure [102]. However, since such an approach does not allow for multiple adversarial updates in one backward step, the authors slightly modified the training procedure. Indeed, the backward step allows for exclusively single-shot – non-iterative – adversarial algorithms. Therefore, in [102], the authors proposed to train the target network on the same minibatch multiple m times in a row. In addition, the number of epochs is divided by m such that the overall number of training iterations remains the same. Finally, once perturbed the minibatch, the perturbation of such is used as a starting-point to perturb the next minibatch.

Finally, the Ensemble Adversarial Training (ETA) framework [103] decoupled the adversarial generation process from the model training one. Indeed, the training dataset of the target model is augmented with adversarial examples crafted by other pre-trained models, by leveraging the transferability property.

Defensive Distillation

The defensive distillation originally refers to the distillation-training technique, whose intuition was first suggested in [104] and then formally proposed by Hinton *et al.* in [105]. Such a training procedure involves transfer the knowledge – the generalization ability – from a bigger model (*teacher*) to

a smaller one (*student*, or distilled model) that could be easily deployed in small devices, e.g., smartphones. In specific, either both the models use the same train set, or the student use a smaller "transfer set". However, as training-objective, the student leverage both the original label of the training sample, and the soft labels (the output vectors ρ) generated by the teacher, with a total loss that combines both of them. For the loss related to the soft labels, the Kullback-Leibler (KL) divergence is exploited [6].

Papernot *et al.* [106] first introduced the defensive distillation. The basic idea was to extract the knowledge from a DNN to improve its own robustness to adversative attacks, by reducing the variations around the inputs, i.e., the network gradients that an adversary can leverage to craft malicious samples. The authors first defined the notion of robustness. A robust DNN should classify inputs in a neighborhood of a given input sample relatively consistently with the latter. Such a neighborhood includes the adversarial version of an input sample, that are L_p-close to that sample.

Adversarially Robust Distillation (ARD) [107] combines the defensive distillation to the adversarial training. Indeed, such a framework leverages the PGD attack to adversarially-train a network, as in [22]. Subsequently, the knowledge of such a network is used to distill a smaller student, as in [105]. In such a way, authors of [107] solved several problems. The adversarial robustness is typically preserved under knowledge distillation, i.e., a distilled student is as robust to adversarial samples as the teacher. However, (i) Not all the robust teachers transfer the robustness to a distilled student, while ARD guarantees that the robustness is transferred. In addition, (ii) ARD can produce students that are more robust than the teacher. Finally, authors also developed Fast-ARD, that is, a version of ARD that exploits the method from [102] to reduce the overhead of adversarial training.

[6] Please refer to the original work of Hinton *et al.* [105] for further details on distillation.

Input-Transformation Techniques

Such techniques involve the transformation of the input image to improve the robustness against adversarial attacks. Indeed, adversarial attacks are crafted by adding noise to the original image. Hence, a way to defend against adversarial attacks is to denoise the perturbed samples either before submitting them to the target model, or during the submission.

Liao *et al.* [108] proposed High-Level Representation Guided Denoising (HGD), that is, a model trained by setting the loss function to the L_1 distance between the outputs of a layer l of the target model generated with both original and adversarial samples. The authors considered two different layer configurations, which yeld to two different versions of HGD. The first pertains to the layer l being the topmost convolutional layer. In such a configuration, the denoiser is guided by the features extracted by the layer, which provide rich supervised information. The second case involves the layer preceding the softmax one, thereby ensuring that the denoiser is guided by the logits, which directly represent the classification results.

PixelDefend [109] makes small changes to an adversarial image in order to move it back towards the training distribution, i.e., moves the image towards a high-probability region. It starts from the adversarial image, and generates a *purified* image that is within an $\varepsilon_{\text{defend}}$-ball of the perturbed image, by using a greedy technique. The magnitude of $\varepsilon_{\text{defend}}$ is adaptively tuned depending on the probability of the input adversarial image. Indeed, images that have high probability under the training distribution would have a low $\varepsilon_{\text{defend}}$ preventing significant modification. Conversely, low probability images would have a high $\varepsilon_{\text{defend}}$ thus allowing significant modifications.

Finally, there are works that leverage randomization as pre-processing technique in adversarial defense, during their submission into the network. Stochastic Activation Pruning (SAP) [110] drops out nodes in each layer during the forward step. In specific, it prunes subset of activation where the probability of retaining a neuron is proportional to the magnitude of its

activation. Surviving activations are then scaled to preserve the expected feature distribution. On the other side, Xie *et al.* [111] add a random resize layer and a random padding layer to the beginning of the classification section of the target CNN. The first resize layer resizes the image with a low resize-step, while the second padding layer pads zeros around the resized image, in a random manner.

2.4.2 Detection Defense-Methods

Detection methods aiming to detect whereas an input sample is legit or malicious – crafted with adversarial attacks methodologies –.

Several approaches leveraged auxiliary models to detect adversarial examples. Defense-GAN [112] leverages a Wasserstein GAN (WGAN) [7] approach to detect both white-box and black-box attacks. The WGAN is first trained on clean training samples. Subsequently, the generic input image is projected onto the generator $\mathcal{G}$, that will generate a clean image – since it was trained on the clean dataset –, thus denoising an eventual perturbation. Subsequently, the generated image will be fed to the target model.

Conversely, other approaches involve either modifying the input image or looking for the dimensionality of images-space to detect whether it is an adversarial sample or not. Ma *et al.* [114] explored the concepts of Local Intrinsic Dimensionality (LID) and adversarial subspaces to detect adversarial examples. The LID can be used to represent the representational dimension of a subspace in which a data point (either a clean image or an adversarial image) belongs – that is, a measure of the rate of growth in the number of data points as the distance from a reference sample increases –. Indeed, it has been proved that the LID of adversarial samples is far greater than that of clean images, since an adversarial perturbation allow the modification of all data coordinates (pixels) of a sample. Therefore, adversarial samples exploit

[7] A WGAN is a GAN that exploits the Wasserstein distance as loss function, aiming to get a more stable training process. Refer to [113] for details regarding WGANs.

the full degree of freedom afforded by the representational dimension of the data domain. Nonetheless, it is computationally difficult to compute the LID. Hence it is typically estimated from a neighborhood of the target sample. Finally, the authors leveraged the estimated LID to train a classifier.

On the other side, Adversarial Detection method via Disentangling Natural images and Perturbations (ADDNP) [115] exploits the concept of LID and adversarial subspaces to justify the fact that adversarial images are more difficult to reconstruct than clean images, hence simple to detect via the reconstruction discrepancy. ADDNP is a dual-branch framework, and each branch is composed of both an encoder and a decoder. The first branch extracts the representation of a clean image, while the second ones first (i) extracts the representation of the perturbation via the encoder, and then (ii) reconstructs the clean image with the decoder, based on the combined representations of the clean image and the perturbation. The two branches are trained separately. Finally, ADDNP computes the reconstruction discrepancy, which leverages the cosine similarity, between a sample (either clean or adversarial) and its reconstructed version obtained by the dual encoder-decoder scheme. Such a discrepancy acts as a score for adversarial detection.

Ryu *et al.* [116] leverage the entropy and the bit-depth reduction of an image to detect adversarial samples. Indeed, they discovered that the entropy of an adversarial sample is bigger than that of its clean counterpart, due to the perturbation. In addition, the entropy of a clean image in which the bit-depth has been reduced is lower than the entropy of the original clean image. Finally, the change in the entropy of an adversarial example after the bit-depth reduction is larger than that of a clean image after bit-width reduction.

Nesti *et al.* [117] leveraged both digital transformation, e.g., blurring or flipping, and a voting mechanism to detect adversarial images. Given an image x and a trained model f, the author performed a digital transformation on x in order to obtain x', and fed the target CNN with both x and x'. Subsequently, they computed the distance between $f(x)$ and $f(x')$ by using the KL

divergence, as in Equation (2.15).

$$D(f(x), f(x')) = \max\{KL(f(x), f(x')), KL(f(x'), f(x))\} \tag{2.15}$$

Since the KL divergence is not symmetric, authors select the maximum between the two distance due to the fact that the maximum is a more conservative estimate of the distance. In addition, adversarial examples are characterized by more asymmetrical KL divergences than clean images. The distance in Equation (2.15) is subsequently performed by feeding the images to three different CNN, and each compared with a threshold. If after a majority voting the distance results to be greater than that threshold, the input image is considered as an adversarial one. In addition, for adversarial examples that are robust to input transformations, authors propose a defense perturbation. Such a perturbation is actually a mask, computed trough an optimization procedure, that, when applied to the input image, produces a new one that is no longer robust to input transformations.

2.4.3 Defense as Side Effect

This subsection explores two methodologies designed originally to reduce the computational complexity of DNNs but also affect the robustness of them against adversarial attack. Nowadays DNNs are being widespread in edge computing devices, whose computational resource are low. Therefore, there is the need to improve the efficiency of DNNs in order to deploy them on small devices. Indeed, there are several techniques aiming to improve the runtime performance of DNNs, storage efficiency, or reducing energy consumption.

Quantization [16, 118, 119] is a compression technique [120, 121] which reduces the bit precision – bit-width – for the weights and activations of a target DNN. There are generally two categories of quantization [122]: (i) Post-Training Quantization (PTQ) uses full-precision parameters during the training process, and after such a process reduces the precision of the

trained parameters. (ii) Quantization-Aware Training (QAT), on the other hands, assesses the effects of parameter quantization during the training process, leading to better results. However, in this thesis we do not aim to provide a comprehensive overview of network quantization as a whole. Instead, our focus is limited to the role of quantization in the context of adversarial robustness. Please refer to [123, 122, 124] for further details about quantization, from both theoretical foundations and the various technique proposed.

On the other side, the **Approximate Computing (AxC)** design paradigm [125, 126] leverages the presence of error-tolerant code in applications to trade off implementation, storage and/or accuracy for performance or energy gains. In brief, AxC exploits the gap between the level of accuracy *required* by applications or users and that *provided* by the computing system, for achieving diverse optimizations. However, it is first necessary to identify the specific part of the application that can be approximated, such as a portion of the code or data. Subsequently, the approximation can be introduced using various techniques, such as reducing the amount of bits used for representing input data and intermediate operands [127], memorizing results for later reuse [128, 129], or using custom approximate hardware, e.g, adders [130, 131] or multipliers [132, 133]. The latter approach is the most widespread, and leverages hardware components that introduce errors within the target application, but allow for gaining performance. Finally, note that the error introduced by the approximation techniques is often estimated trough simulation-based methodologies. In the field of ML, the AxC design paradigm leverages the inner error-resiliency of DNNs – due to noisy and redundant training datasets, as well as the training procedure – in order to reduce both the computational time and the hardware-resource requirements of the training and inference phases [134]. In the computer vision task, one of the most adopted technique is to replace the multipliers used in the convolutional and fully-connected layers of the CNNs with approximate ones.

Indeed, multipliers within layers are the most demanding component in terms of hardware overhead [135, 136]. However, in this thesis we do not aim to provide a comprehensive overview of AxC as a whole. Instead, our focus is limited to the role of AxC in the field of ML, and in particular in the context of adversarial robustness. Please refer to [135, 137] for further details about approximation of DNNs.

Finally, note that while discussing about quantization and AxC, we will refer to the normal full-precision DNNs as either Full-Precision Neural Networks (FpNNs), or just *normal* DNN. In addition, quantized and approximated DNNs are denoted by Quantized Neural Networks (QNNs) and Approximate Neural Networks (AxNNs), respectively. Finally, we will also denote with *implementation* each version of a DNN, meaning that a DNN could have more than one implementations, e.g., the full-precision, the quantized, and the approximate implementations.

Quantization as a Defense Method

Several works from the scientific literature have analyzed the role of quantization in the context of adversarial machine learning, from both the perspectives of attack [138] and defense. In this section, we will review the role of quantization as a mitigation strategy against adversarial attacks, although the results could be contrasting.

Bernhard *et al.* [139] first analyzed the effects of quantization on adversarial machine learning. Indeed, they discovered that quantization offers poor-protection against SotA adversarial attacks, such as FGSM and ZOO. In addition, the transferability between a FpNNs and a QNNs is poor, since the quantization can map two different values into the same bucket. Finally, the quantization of the activations can lead to gradient masking, making the quantized network robust against gradient based attacks, e.g., BIM or PGD. Conversely, Lin *et al.* [140] found that QNNs are less robust to the FGSM attack than their full-precision counterpart, due to the *error amplification*

effect, discussed by Liao *et al.* in [108]. Therefore, authors of [140] proposed a method to increase the robustness of QNNs against adversarial attacks, by suppressing the error amplification effect. Duncan *et al.* [141] also confirmed that the low transferability from FpNNs to QNNs. However, they found that a QNN is *relative robust* to its full precision network. This means means that if a FpNN is robust against adversarial attacks, its quantized counterparts remains robust against the same attacks. Gorsline *et.al* further explored the relationship between adversarial attacks and quantization in [142]. One one hand, quantizing a network with a higher bit-width for parameters allows the network itself to define tighter decision boundaries between classes. On the other hands, reducing bit-width increases the average distance between test data and decision boundaries, which increases the amount of adversarial perturbation needed to case the input to change class. In addition, authors discovered that there is a critical attack strength that balances the above effects. Finally, Li *et al.* [143] examined precisely the quantization-adversarial robustness by specifying both whether the PTQ or the QAT is selected, and which parameters are quantized (weights and/or activation). They found that both PTQ and QAT QNNs exhibit an increase in robustness, w.r.t. their full-precision counterpart, with the reduction of quantization bit-width, if the adversarial strength is lower than a threshold. Also, activation quantization enhances robustness more noticeably compared to weight quantization. Conversely, if the quantization is combined with robust parameters, such as the adversarial training, (i) PTQ causes a decreasing in both the robustness and the precision of the QNN, w.r.t. the FpNN, as the quantization bit-width decreases. Instead, (ii) QAT increases the accuracy, and increase the robustness of against adversarial attack if combined with adversarial training, although introducing a training overhead.

Approximate Computing as a Defense Method

The role of AxC in adversarial machine learning in the domain of computer vision is an intriguing subject that merits further exploration. However, the research community has only recently begun to examine this topic, with only three papers analyzing it between 2021 and 2022, and results being contrasting.

Guesmi *et al.* [144] were the first to explore the AxC in the field of adversarial machine learning, by (i) replacing the multiplier used for convolutions within convolutional layers with an approximate 32-bit floating-point one, and (ii) evaluating the robustness of the resulting neural network against both white-box and black-box adversarial attacks. In specific, authors replace exclusively the conventional mantissa multipliers, since it has been shown that introducing errors in the exponent part might have drastic impact on the accuracy of CNNs [145, 146]. As a result, the crafted AxNN leverages an approximate convolution function. Once approximated the target network, the authors both explored the transferability from the FpNN to the approximate version, and attack directly the AxNN, with gradient-based (white-box), score-based and decision based (black-box) attacks. They found that the transferability is poor from FpNNs to AxNNs, resulting in AxC being a mitigation method against transfer-based attack. Similarly, the success-rate of black-box attack is poor, always below 30%. Conversely, white-box attacks succeed to fool the AxNN, depending on how big is the perturbation applied. Although, the perturbation required to fool an AxNN is bigger to that required to fool the FpNN, resulting in the former being more robust against white-box attack than the latter.

Siddique and Hoque [147] generated approximate networks by replacing their multipliers with integers one from the EvoApprox8b library [148]. For what pertains to white-box attacks, authors observed that, although being more robust than a FpNN, the robustness of a AxNN decreases with an increase in the strength of the adversarial attack (how much perturbation is applied to

the input image). In addition, both the FpNN and its approximate counterpart exhibit more robustness against L_2 attacks than L_∞ ones. Furthermore, in contrast to [144], authors found that AxNNs are vulnerable to gradient-based attacks even tough the perturbation applied is small. Finally, authors compared the robustness of AxNNs and 8-bit QNNs, in specific by approximating quantized networks. They found the AxC acts antagonistically to quantization w.r.t. adversarial attack. This means that, while the quantization improves the robustness of FpNN against adversarial attack, the AxC decreases the robustness of QNNs.

Finally, Atoofian [149] proposed to approximate a FpNN with an ensemble of approximate multipliers in order to increase its robustness to adversarial attack. However, although the author claimed to use an ensemble of approximate multipliers to generate the AxNN, he actually used a single multiplier composed of $1+P$ compressors [8], an exact one and P approximate ones. The compressor to use during the multiplication is selected by a multiplexer. Subsequently, the author generated n approximate versions of the same target DNN, and each AxNN selected a different compressor to use within the same multiplier. Finally, all the n AxNNs are fed with the same inputs, and the final output – classification – is obtained by either averaging or maximum voting the n outputs generated by the AxNNs. Hence, in this work he used an *ensemble approximate network* to enhance the robustness. Atoofian then targeted with both white-box and black-box attacks the *ensemble approximate network*, and compared results w.r.t. the FpNN, an AxNN obtained with a single approximate multiplier, and two QNNs, the former when only weights were quantized, the latter with both weights and activations quantized. Results show that the *ensemble approximate network* is always more robust to adversarial attacks than other implementations, except of the QNN with

[8] A compressor within a multiplier reduces the number of bits outputted by the multiplier itself. Please refer to [150] for details regard approximate multipliers generated by suing an ensemble of compressor.

quantized weights only. Indeed, the robustness of such a quantized version is almost the same as the *ensemble approximate network*.

2.4.4 Challenges

The preceding sections reviewed defense techniques against evasion adversarial attacks, w.r.t. several implementations of DNNs. Nevertheless, there are as yet unresolved challenges that this thesis must confront, especially pertaining to efficiency techniques that affect the robustness to adversarial attacks

Challenge 1 QNNs should be more robust against adversarial attack than their full-precision implementation. However, results from the scientific literature are either partial, not clear, or contrasting [139–141].

Challenge 2 It has been proved that the AxC design paradigm can improve the robustness of DNNs against adversarial attacks. However, although an interesting research topic, especially pertaining to edge computing in the image classification tasks, only three papers [144, 147, 149] were proposed in the scientific literature. In addition, such works do not cover other approximation techniques that could enhance the robustness, e.g., using different approximate multipliers in the same networks. Therefore, this research area needs further investigation.

Challenge 3 Although QNNs and AxNNs are more robust against adversarial attack thank FpNNs, they are still vulnerable to them. The scientific literature lacks of paper that both try to defend such networks against adversarial attack, and that explore the possibility to use defense techniques – originally designed for FpNN – to defend such implementations.

Challenge 4 Although adversarial attacks based on quantization have been explored in the literature [151, 138], no prior work has employed the AxC to design such attacks.

3

Improving the Stealthiness of Adversarial Attack against CNNs

This Chapter presents and discusses two adversarial attack methodologies that we proposed, named N-Pixels (NPs) [68], and Incremental-Pixels (IPs). We will provide a comprehensive overview of the main steps involved in all the three attack-methodologies, and the manner in which they facilitate the resolution of challenges posed by adversarial attacks. In addition, we will present exhaustive experimental evaluations of the proposed attacks to demonstrate their suitability and applicability in real-world scenarios.

As discussed in the Section 2.2 of the previous Chapter, Convolutional Neural Networks (CNNs) are vulnerable to adversarial attacks. They involve the creation of perturbed, yet legit, inputs that are visually similar to the original inputs but can force a trained model to produce incorrect outputs. Such attacks are designed to target the model either in the (i) training phase, in order to corrupt the functionalities of the target model – poisoning attacks –, or in the (ii) testing phase, aiming to gain knowledge about the target model – exploratory attacks – or to cause a misclassification of a perturbed input sample – evasion attacks –. Furthermore, for what pertains to the knowledge required to conduct such attacks, white-box attacks require full access to the model (such as the architectural details, internal parameters and training-

procedure). On the other side, gray-box attacks require partial knowledge about the model. Finally, black-box attacks assume no knowledge about the model, requiring access exclusively to the inputs and outputs of the model.

Several adversarial attack methodologies were proposed in the scientific literature, although there are several challenges, discussed before, that need to be confront. Indeed, white-box attacks are unrealistic and not-applicable in real world scenarios (Challenge 1). Furthermore, black-box attacks are typically computationally-intensive or still require access to information that should be not available – score based attacks – (Challenge 2). In addition, evasion attacks (both the white-box and the black-box scenarios) (i) introduce perturbations that are still visible in the final adversarial images (Challenge 4), (ii) perturb pixels within the input images that are not important for the final classification – Challenge 3–, and (iii) they suffer from non-reproducibility, since they leverage stochastic methodologies to compute malicious samples (Challenge 5).

NPs and IPs contribute to face this challenges. In specific:

- NPs is a gray-box attack that requires less information than white-box ones. Indeed, such an attack leverages exclusively the information about the architecture of the target model, without knowledge about the training procedure. In addition, such an attack craft adversarial images by perturbing only the sections within the original input image that are significant to the final classification of the model. Finally, NPs leverages deterministic methodologies to introduce a not-visible perturbation within the input image.

- IPs improves the One-Pixel (OP) attack by (i) perturbing a number of pixels that is not fixed priory. Instead, IPs tries to craft an adversarial images by iteratively modifying a growing number of pixels. Furthermore, such an attack (ii) leverages deterministic methodologies to introduce an imperceptible perturbation within the input image. The

process of computing the perturbation for a specific pixel involves an analysis of the pixels surrounding the target one.

3.1 N-Pixels: a Novel Grey-Box Adversarial Attack for Fooling Convolutional Neural Networks

NPs is a gray-box adversarial attack that contribute to the State-of-the-Art (SotA) by proposing solutions to several challenges. Indeed, it requires access to the architecture of the target model in order to craft adversarial images, without needing other information such as the training procedure, in response to the Challenge 1. Furthermore, NPs craft the adversarial images by modifying exclusively the pixels that are important to the final classification and hence by addressing the Challenge 3. Finally, it leverages deterministic approaches for computing and applying the perturbation to the pixels, allowing to both reproduce the attack (in response to the Challenge 5) and introduce invisible perturbation, in response to the Challenge 4. In the following, we will detail how NPs addresses the mentioned challenges, and we will present experimental results that prove the suitability of the attack.

3.1.1 Methodology

NPs is divided in two different stages, aiming in modifying the input image as few as possible. The first stage involves identifying the section within the input image that contributes the most to the classification of the image itself. In particular, the attack examines the output of the final feature-layers of the feature learning section of the CNN to identify the most significant feature that contributes the most to the classification. Subsequently, such feature is mapped in reverse to the input image in order to identify the section of the image related to the feature deemed important. Finally, in the second stage, a binary search algorithm is employed to identify an appropriate – and minimal – perturbation to be applied to the section of the image identified in the previous stage.

The following Subsections will provide a comprehensive examination of each stage.

First Stage: Identification of the Most Important Section within the Input Image

In the first stage of the NPs attack, the objective is to identify the sections of the input image that are of greatest significance, by leveraging the information related to the architecture of the target CNN. Such a stage allow us to address the Challenge 3.

Assume that such model has L layers $l_i, i \in [0,\dots,L-1]$. l_0 is the input layer, followed by Z features layers $l_i, i \in [1,\dots,Z]$ – with $Z < L$ – of the feature learning section. Finally, there are the classification layers $l_i, i \in [Z+1,\dots,L-1]$, with l_{L-1} the softmax layer.

Identification of the most important feature In order to identify the most significant feature that contribute the most to the classification of the input image x, the proposed attack analyzes the output of the last feature layer l_z. The reason is that, as the depth of the CNN increases, the features extracted by the layers become progressively more representative of the image. Consequently, deeper layers capture more abstract and high-level features, which significantly contribute to the accuracy and effectiveness of the classification process [152].

First, NPs lets the target model to perform an inference on the input image, x, with the objective of conducting an analysis of the layer l_z. In this context, we denote the input and the output volumes, of the layer l_z, with I_z and G_z, respectively. In particular, G_z is a volume composed by $w \cdot h$ fibers, that are vectors whose dimension is k. The width w and height h are dependent on the number of filters, the stride, the padding, and the dimension of the input, while k is the number of filters. Finally, each fiber f belonging to G_z is associated with a specific section S_z of the input volume I_z. Consequently,

the identification of the most significant fiber $f^* \in G_z$ allows for identifying the most important section S_z^* within the input I_z. Figure 3.1 illustrates the aforementioned discussion in graphical detail.

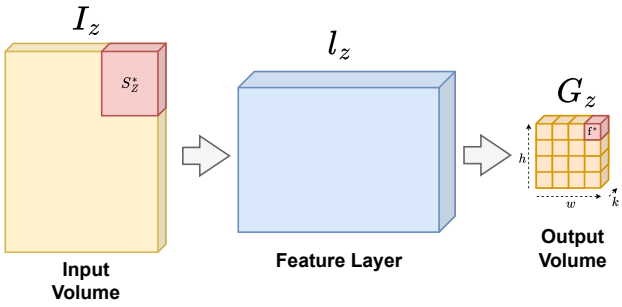


Figure 3.1: Representation of input and output volumes I_z and G_z, respectively, of a feature layers l_z. In this context, f^* is the most important fiber within G_z, that corresponds to the most important section S_z^* of the input volume.

The procedure 1 illustrates the method for identifying the fiber f^*, that will contribute the most to the final classification, between two fibers, $f^1, f^2 \in G_z$, where $f^1 \neq f^2$. The number of values of the fiber f^1 that are equal to or greater than the values of f^2 in homologous positions (lines 4 to 10) is counted, in order to determine whether the fiber f^1 is *better* of the fiber f^2. Given two sections S_z^1 and S_z^2, within the input volume I_z of the layer L_z, the neurons of such layer compute the fiber f^1 and f^2 of the output volume G_z by leveraging the k filters. If $f^1(i) > f^2(i)$, $i \in [0, \ldots, k-1]$, this means that the section S_z^1 is *more important* of the section S_z^2, according to the filter i. The reason is that the Rectified Linear Unit (ReLU) and its variants are the most widely used activation functions. Thus, the largest is a value within a fiber f, according to a certain filter i, the most it will contribute in the subsequent layer [153, 154]. Such reasoning is repeated for each fiber f within the volume G_z.

Back-mapping Once identified the most-important fiber f^*, NPs leverages the stride, the padding, and the number of filters in order to search the exact

Algorithm 1 Procedure to find the fiber that contribute the most to the classification between two given fibers.

```
procedure FINDBESTFIBER(f^1, f^2, k)
    α_1 ← 0
    α_2 ← 0
    for i ← 0; i < k; i ← i + 1 do
        if f^1(i) ≥ f^2(i) then
            α_1 ← α_1 + 1
        else
            α_2 ← α_2 + 1
        end if
    end for
    if α_1 ≥ α_2 then
        f* = f^1
    else
        f* = f^2
    end if
    return f*
end procedure
```

position of the most-important sections S_z^* within the input volume I_z of l_z. Of course, the volume I_z is also the output volume G_{z-1} of the layer l_{z-1}. Therefore, we map the most important section $S_z^* \in I_z = G_{z-1}$ through the layer l_{z-1}, in order to identify the most important section S_{z-1}^* within the input I_{z-1} of the layer l_{z-1} that generated the most important section S_z^* (set of fibers) within the output volume G_{z-1}.

This process is repeated until a layer, designated as layer l_a, is reached, as depicted in Figure 3.2. Such a layer is situated in proximity to the input layer. Note that the selection of the stopping layer is contingent upon the desired precision of mapping relevant features. The closer the selected layer, l_a, is to the input image, x, the more precise the mapping of relevant features will be. However, this increases the computational complexity. Accordingly, the optimal selection of the layer l_a entails a trade-off between the computational complexity and the precision of the mapping.

Consequently, at the conclusion of the back-mapping procedure, the most significant section, denoted by S_a^*, is identified within the input volume, I_a,

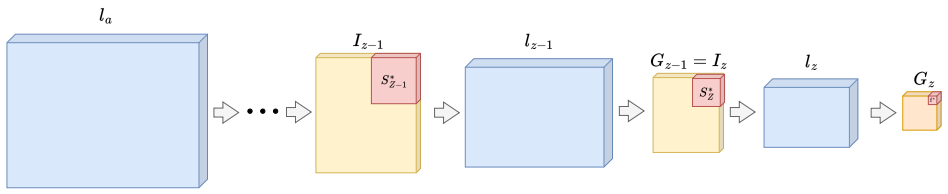


Figure 3.2: The process of back-mapping. The most important section S^*_z within the volume I_z, identified through the most important fiber f* in the volume G_z, corresponds to a set of fibers within the output volume G_{z-1} of the layer l_{z-1}. These fibers are back-mapped in order to identify the section S^*_{z-1} within the input volume I_{z-1}. The entire process is repeated up to the layer l_a

of the layer, l_a. Finally, by leveraging the information related to the stride, padding, and filter dimensions, NPs map the S^*_a directly to the input image x. This process allows identifying the most important section S^* of the input image.

Second Stage: Crafting the Adversarial Image

Assume that the input image can be expressed by $x = \{x_0, \ldots, x_{H-1}\}$, with H denoting the number of pixels. Once identified the important section $S^* \in x$, composed by P pixels $x_j, j \in [0, \ldots, P-1]$, with $P \leq H$, the second stage craft the adversarial image by applying to such P pixels a perturbation δ_x.

In order to comprehend the functioning of this stage, please refer to Table 2.1, and Equations (2.4) and (2.2). The second stage of the NPs attack modifies the pixels within the section S^* by applying and evaluating an additive perturbation δ_x to them. In specific, the second stage chooses a perturbation δ_x, and, iteratively, either increments or decreases it, in order to (i) craft a successful adversarial image, (ii) that is as similar as possible to the original image.

For each pixel j, the specific perturbation δ_x applied to it is evaluated with two distinct distance metrics.

- The attribute v_j represents the extent to which ρ^* – the likelihood that the image belongs to the correct class y –, is altered by modifying the pixel j of the input image x with the perturbation δ_x. Once modified the jth pixel by adding the perturbation δ_x, the resulting image can be expressed $\hat{x} = \{x_0, \ldots, x_j + \delta_x, \ldots, x_{H-1}\}$. Consequently, the classification vector assigned to $\hat{x}$ is expressed as $\hat{\rho} = \{\hat{\rho}_0, \ldots, \hat{\rho}_{M-1}\}$ (see Table 2.2), with $\hat{\rho}_i$ the likelihood that the perturbed image $\hat{x}$ belongs to the generic class y_i. Finally, we define with $\hat{\rho}^*$ the likelihood that $\hat{x}$ belongs to the correct class y. Consequently, the value v_j is computed as in Equation (3.1):

$$v_j = \texttt{dist}(\rho^*, \hat{\rho}^*) \tag{3.1}$$

  In the event that the modification of the single pixel j causes a misclassification, such that $argmax(\rho) = y$, and $argmax(\hat{\rho}) = \hat{y}_a$, with $\hat{y}_a$ a different class from the correct one y, the attribute v_j is assigned the maximum value (Ω_v). Therefore, such a value represents how far the perturbed image $\hat{x}$ is moving away from being correctly classified, without preferring a specific class different from the original one (NPs is an untargeted attack).

- The weight w_j, represents the extent to which the new image $\hat{x}$ differs from the original image x, once applied the perturbation δ_x to the pixel j. Such a weight is computed as in Equation (3.2).

$$w_j = \texttt{dist}(x, \hat{x}) \tag{3.2}$$

  Several distance-metrics can be adopted, such as the norm (either L_0, L_2, or L_∞), the Structural Similarity Index Measure (SSIM), or the Peak Signal-to-Noise Ratio (PSNR). It should be noted that a maximum distance value, denoted as Ω_w, is permitted in order to ensure that

the perturbed image $\hat{x}$ is as similar as possible to the original image x. Accordingly, in order to generate a valid adversarial image, it is necessary to adhere to the constraint $w_j \leq \Omega_w$, denoted as *similarity constraint*, for each pixel that has undergone modification through the application of the perturbation δ.

Therefore, since NPs evaluates the application of a specific perturbation δ_x for all the P pixels of the section S^*, for each perturbation we have P attributes v, and P weights w. The Procedure 2 outlines the methodology for the evaluation of the perturbation δ_x applied to the pixels of the section S, based on the values of v and w.

Algorithm 2 Procedure for evaluating how the perturbation of a single pixel affect the image.

procedure PERTURBATIONEVALUATION(x, ρ^*, y, P, δ_x, Ω_v, Ω_w)
  $T \leftarrow \emptyset$
  for $j \leftarrow 0; j < P; j \leftarrow j+1$ **do**
    $v_j \leftarrow 0$
    $w_j \leftarrow 0$
    $\hat{x} \leftarrow x[j+\delta_x]$
    $w_j \leftarrow \texttt{dist}(x, \hat{x})$
    if $w_j > \Omega_w$ **then**
      $w_j \leftarrow -1$
    else
      $\hat{\rho} \leftarrow f(\hat{x})$
      $\hat{y}_a \leftarrow \texttt{argmax}(\hat{\rho})$
      if $\hat{y}_a \neq y$ **then**
        $v_j = \Omega_v$
      else
        $v_j \leftarrow \texttt{dist}(\rho^*, \hat{\rho}^*)$
      end if
    end if
    $t_j \leftarrow \frac{v_j}{w_j}$
    $T \leftarrow T \cup t_j$
  end for
  $T \leftarrow \texttt{DescendingSort}(T)$
  return T
end procedure

However, the perturbation δ_x has first to be computed, in order to be applied to the pixels of S^* and evaluated. In essence, our approach entails an iterative computation and updating of the perturbation δ_x, with the objective of identifying a perturbation that allows for the generation of an adversarial image that is as similar as possible to the original image, by leveraging a bisection method into the range $[\gamma, \Gamma]$ – hence by addressing the Challenge 5 –. The initial perturbation is computed as in Equation 3.3, and updated by modifying the values of either γ or Γ.

$$\delta_x \leftarrow \left\lfloor \frac{\Gamma + \gamma}{2} \right\rfloor \tag{3.3}$$

The procedure 3 depicts how to compute and evaluate the most suitable perturbation, and how to craft the adversarial image. The initial perturbation is computed as in Equation 3.3, and evaluated with the procedure 2 (lines 8 and 9). Consequently, by leveraging the ordered set T of ratios v/w, we perturb the original image x by adding the perturbation δ_x to the P pixels, starting from the best ones (the ones whose ratio v/w is bigger). There are three different cases to analyze.

1. Once modified a pixel, the *similarity constraint* is violated. Consequently, a smaller perturbation is computed by updating $\Gamma = \delta_x - 1$, and evaluated again with the procedure 2 on all the P pixels (line 15).

2. The modification of that pixel generates an adversarial image ($\hat{x}$ is misclassified). Therefore, a smaller perturbation is computed by updating $\Gamma = \delta_x - 1$, and evaluated again with the procedure 2, in order to try to craft a new adversarial image that is more similar to the original image than the current adversarial image.

3. Neither the *similarity constraint* has been violated, nor the perturbed image $\hat{x}$ causes a misclassification. Therefore, the perturbation δ_x is also applied to the pixel whose ratio v/w is the following in the set T,

in order to compute a new perturbed image, that has more than one pixel perturbed with the same perturbation δ_x.

If the case 3 occurs while applying the perturbation δ_x to all the P pixel, a bigger perturbation is computed by updating $\gamma = \delta_x + 1$ end evaluated once more. The process of computing and evaluating a procedure stops when the range $[\gamma, \Gamma]$ is smaller than a threshold μ. Finally, the procedure 3 returns the adversarial image only if the case 2 occurs while evaluating a perturbation δ_x and applying it to the pixels of the section S^*. Hence, the NPs attack succeeds. Otherwise, if cases 1 and 3 occur for all the computed perturbations, the procedure 3 returns the original image itself, resulting in the failure of the NPs attack for that image. Finally, note that both cases 1 and 2 try to compute a new adversarial version of x by reducing the perturbation, hence by addressing the Challenge 4.

3.1.2 Experimental Evaluation

In order to evaluate the NPs attack, we targeted different CNNs and datasets from the scientific literature. The following Sections provide the reader with details related to the experimental setup, and the analysis of the experimental results. Finally, we compare our attack with other SotA attacks.

Setup

In this Subsection, we provide the reader with details related to the CNNs and datasets we exploit to evaluate our attacks, and how we configured NPs attacks in both the first and the second stage.

Networks and Datasets We consider several datasets and different CNNs. In particular, we targeted two networks from the ResNet family [155], i.e., ResNet24 and ResNet50, trained to cope with the CIFAR-10 [156] and Dogs vs. Cats [157] datasets, respectively. Finally, we targeted MobileNet-v3 [158]

Algorithm 3 Procedure for selecting a perturbation to evaluate, and subsequently crafting the adversarial image.

procedure CRAFTADVERSARIALSAMPLE(x, S^*, Ω_v, Ω_w, Γ, γ, P, μ)
  $x^{adv} \leftarrow x$
  $\hat{x} \leftarrow x$
  $\rho \leftarrow f(x)$
  $y \leftarrow \texttt{argmax}\{\rho\}$
  $\rho^* \leftarrow \texttt{max}\{\rho\}$
  while $\Gamma - \gamma \geq \mu$ **do**
    $\delta_x \leftarrow \left\lfloor \frac{\Gamma+\gamma}{2} \right\rfloor$
    $T \leftarrow \texttt{PerturbationEvaluation}(\, x, \rho^*, y, P, \delta_x, \Omega_v, \Omega_w)$
    for $t_i \in T$ **do**
      $\frac{vj}{wj} \leftarrow t_i$
      $\hat{x} \leftarrow x[j + \delta_x]$
      $d \leftarrow \texttt{dist}(x, \hat{x})$
      if $d > \Omega_w$ **then**
        $\Gamma \leftarrow \delta - 1$
        Break
      end if
      $\hat{\rho} = f(\hat{x})$
      $\hat{y}_a \leftarrow \texttt{argmax}(\hat{\rho})$
      if $\hat{y}_b \neq y$ **then**
        $\Gamma \leftarrow \delta - 1$
        $x^{adv} \leftarrow \hat{x}$
        Break
      end if
    end for
    $\gamma \leftarrow \delta_x + 1$
  end while
  return x^{adv}
end procedure

on the FGCV7 dataset [159]. Please note that both ResNet50 and MobileNet-v3 were pre-trained on the ImageNet datasets [55] dataset, and then fine-tuned on the Dogs vs. Cats and FGCV7 datasets, respectively. Table 3.1 reports the accuracy of each CNN on the respective test dataset.

	ResNet24	**ResNet50**	**MobileNet-v3**
Accuracy (%)	86.39	97.01	98.08

Table 3.1: Success-rate of ResNet24, ResNet50, and MobileNet-v3 on the CIFAR-10, Dogs vs. Cats, and FGCV7 test datasets, respectively.

CIFAR-10 consists of 60 thousand 32×32 RGB images, each belonging to ten different classes, e.g., ship, car, and frog, while the Dogs vs. Cats datasets consist of 25 thousand 224×224 RGB images belonging to the classes *cat* and *dog*. Finally, the FCGV7 dataset is composed by 3650 224×224 high-quality RGB images of multiple real-life symptoms of diseases affecting plants. The datasets were partitioned in order to allocate 80% of the images for training purposes and reserve the remaining 20% for use as test dataset. Finally, 15% of the training dataset was employed for validating the training process, serving as a validation dataset. All the CNNs we mentioned have been trained using cross-entropy as loss functions, Adam as optimizer, and ReLU as activation function.

N-Pixels: First Stage Configuration For what pertains to the first stage of the NPs attack, the search of the most important output fiber f^*, within the last feature layers, was implemented as specified in the procedure 1. Once identified f^*, it was back-mapped up to the layer L_a. We have chosen the first, and the second, convolutional layer that followed the input layer for what pertains to the ResNet50 and MobileNet-v3, respectively. Conversely, regarding ResNet24, we selected the fifth convolutional layer as l_a. The reason is that, since the input images of both ResNet50 and MobileNet-v3

are RGB 224×244 images, in order to better identify the important section within such images, a greater precision is required than the case of ResNet24.

Furthermore, assume that NPs targets a test image x (from one of the three datasets we considered) in input to a CNN (either ResNet24, ResNet50 or MobileNet-v3). In order to reduce the computational cost, once back-mapped the most important fiber up to the layer l_a, and identified the section S_a^*, NPs did not map such section entirely to the input image. Instead, we applied the K-Means clustering algorithm in order to cluster the section S_a^* in different K clusters $S_i \in S_a^*$, with $i \in [0,\ldots,K]$. Among these clusters, we selected the biggest one, i.e., the one whose number of important elements is greater than the others, namely, S_{ak}^*. Thus, within an input image x, NPs identified a subsection S_k^* within the most important section S^*.

N-Pixels: Second Stage Configuration Once identified the subsection S_k^* within an input image x, NPs perturbs it as depicted in Procedure 3, in order to select a perturbation, and evaluate it on the pixels of S_k^*. Since NPs targeted RGB images, each pixel of the image is a tuple of three channels $[r,g,b]$, whose values $r,g,b \in [0,255]$ are represented in 8-bit integer. Thus, the minimal perturbation that can be applied to a channel is $\gamma = 1$, while the maximum perturbation is $\Gamma = 255$. Consequently, the initial perturbation, computed as in Equation (3.3), is $\delta = \left\lfloor \frac{255+1}{2} \right\rfloor = 128$ (line 8 of procedure 3). By evolving the binary search algorithm, the next value was selected between either $\delta_x = \left\lfloor \frac{255+128}{2} \right\rfloor = 192$ or $\delta_x = \left\lfloor \frac{127+1}{2} \right\rfloor = 64$, in order to increase or decrease the additive perturbation, respectively (lines 15, 21, and 26 of the procedure 3). The binary search algorithm, and thus the updating of the perturbation, was performed until $\Gamma - \gamma = 1$, thus, $\mu = 1$. Once selected a perturbation δ, NPs evaluated it by leveraging Equations (3.1) and (3.2). In specific, consider an image x from the test-set, and the perturbed image $\hat{x}$ crafted by applying the perturbation δ_x to the pixel j. Furthermore, $\rho_a = \rho^* = \texttt{argmax}\{f(x)\}$ and $\hat{\rho}_a = \hat{\rho}^* = \texttt{argmax}\{f(\hat{x})\}$ are the likelihoods that x

and $\hat{x}$ belong to the correct class y, respectively, NPs computes the attribute v_j and the weight w_j as follows:

- $v_j = |\rho^* - \hat{\rho}^*|$
- $w_j = 100 - \texttt{PSNR}(x, \hat{x})$

The PSNR measures the difference between two different images, in terms of noise. The smallest is this value, the more visible is the noise. Generally, a PSNR greater than 30 dB means that the difference between two image is almost imperceptible to human eye. Therefore, we choose as $\Omega_w = 70$ as a threshold for the *similarity constraint*. Finally, Ω_v was selected to be equal to 1. Note that, once selected a perturbation δ_x, which is always positive by construction, NPs evaluated both the additive and the subtractive perturbation. Finally, the proposed attack evaluated the application of such perturbation on either a single channel, two channel, or all the three channels, for each pixel.

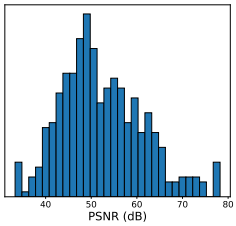
Result Analysis

In this Section, we analyze the experimental results. In specific, Table 3.2 depicts the success-rate of the NPs attacks against ResNet24, ResNet50, and MobileNet-v3, while Figure 3.3 depicts how the PSNR is distributed once crafted successful adversarial image. Note that, in order to conduct such experiments, we make NPs undergoes the similarity constraint of $\Omega_w = 70$ (the minimal PSNR was 30 dB).

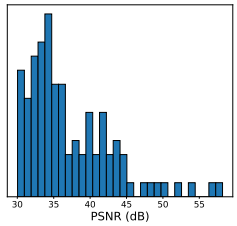
	ResNet24	**ResNet50**	**MobileNet-v3**
NPs	91.9	22.01	89.17

Table 3.2: Success-rate of NPs against ResNet24, Resnet50, and MobileNet-v3.

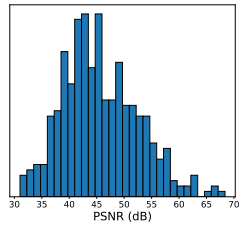
As Table 3.2 depicts, the success-rate of NPs is about 90% against both ResNet24 on CIFAR-10, and MobileNet-v3 on FCGV7, while it is slightly



(a) ResNet24.



(b) ResNet50.



(c) MobileNet-v3.

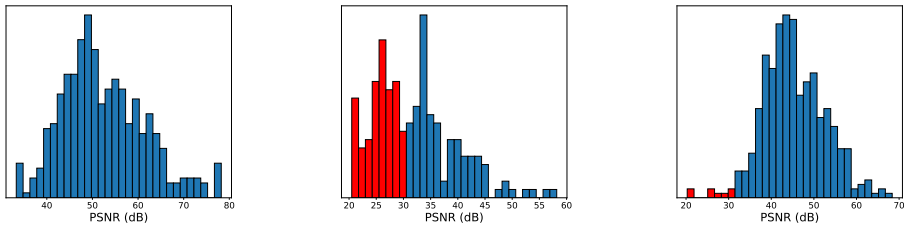
Figure 3.3: Distribution of the PSNR while targeting ResNet24 (**a**), ResNet50 (**b**), and MobileNet-v3 (**c**) on the CIFAR-10, Dogs vs. Cats, and FGCV7 datasets, respectively.

more than 20% while targeting ResNet50 on Cats vs. Dogs. These results are consistent with the ones showed in Figure 3.3. In specific, once crafted successful adversarial samples, we measure the difference between the original image and its adversarial counterpart with the PSNR, for all the images of each test set. Note that, the majority of PSNR-values are distributed around 42-43 dB in case of MobileNet-v3, and around 50 dB in case of ResNet24. Conversely, the values of PSNR are distributed around 30-35 dB in the case of ResNet50. This means that both N-Pixel have a high success-rate against ResNet24 and MobileNet-v3, and, also the adversarial images result to be indistinguishable from the original images, to the human eye. Conversely, the success-rate is lower while targeting ResNet50, and the perturbation introduced by NPs to fool such a network is greater than those introduced to fool ResNet24 and MobileNet-v3.

Given the suboptimal results observed in the ResNet50 case, we re-executed the experiments, this time relaxing the similarity constraint. In specific, we have selected the minimum PSNR equal to 20 dB, hence the threshold of the *similarity constraints* was $\Omega_w = 80$. Table 3.3 show the success-rate of NPs while relaxing the *similarity constraint*, while Figure 3.4 shows results while relaxing the *similarity constraints*.

	ResNet24	ResNet50	MobileNet-v3
NPs	91.9	40.09	90.03

Table 3.3: Success-rate of NPs against ResNet24, Resnet50, and MobileNet-v3. In this case, the *similarity constraint* was relaxed, by setting a minimal PSNR to 20 dB.

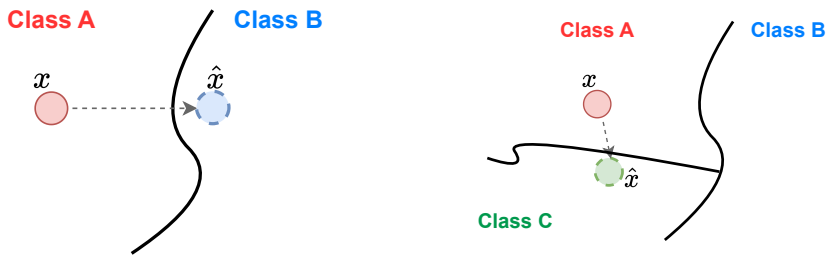


(a) ResNet24. **(b)** ResNet50. **(c)** MobileNet-v3.

Figure 3.4: Distribution of the PSNR while targeting ResNet24 (**a**), ResNet50 (**b**), and MobileNet-v3 ((**c**) MobileNet-v3.) on the CIFAR-10, Dogs vs. Cats, and FGCV7 datasets, respectively. In red, there are additive results obtained by relaxing the *similarity constraint*, with a minimal PSNR to 20 dB.

Note that, while targeting ResNet24 and MobileNet-v3, both the success-rate and distribution of values of PSNR remain largely unchanged. As Table 3.3 shows, the success-rate remains unaltered while targeting ResNet24, and increases by less than 1% while targeting MobileNet-v3. Furthermore, Figure 3.4 **(c)** shows that there are very few adversarial images whose PSNR, w.r.t. their original counterpart, is in the range $[21, 30]$ dB (red results), while results regarding Resnet24 remain unchanged (Figure 3.4 **(a)**). Conversely, for what pertains to ResNet50 on Dogs vs. Cats, the results have changed drastically. Indeed, the success-rate has almost doubled and, as Figure 3.4 **(b)** depicts, the number of PSNR-values between 20 and 30 dB are almost equal than the values greater than 30 dB (red results). The reason of such behavior depends on both the dataset and how we evaluate the perturbation on a pixel, with the ratio v/w. When the ratios are sorted in descending order (the set T, as in line 22 of procedure 2), the NPs attack modifies the pixels in a manner

that causes the image to deviate from the correct classification, without preferring any particular incorrect class. Furthermore, the attack aims to introduce as less perturbation as possible, with the *similarity constraint*.



(a) Binary classification.

(b) Multiclass classification.

Figure 3.5: Example of how NPs craft an adversarial image $\hat{x}$ from the original one x in the case of binary classification problem **(a)**, and a multiclass classification problem **(b)**.

As Figure 3.5 depicts, creating an adversarial image is more simple –in terms of additive perturbation – in the case of a multiclass problem, rather than in the case of a binary problem, without relying on a full knowledge of the target model. The reason is that NPs moves the image x towards the nearest decision boundary with the objective of perturbing it as little as possible, while ensuring the that the *similarity constraint* is met. Assume that the model is quite sure that x belongs to the (correct) class A, meaning that, if $\rho = f(x)$, and $y_a = y = \texttt{argmax}(\rho) = A$, the likelihood $\rho_a = \rho^* = \texttt{max}(\rho)$ that the image belongs to the correct class A is near to 1. In the binary classification problem (Figure 3.5 **(a)**), moving away x from its correct class means preferring the only other class, the class B. Since x is far from the decision boundary, a substantial degree of perturbation is required to effect a change in classification. Conversely, in a multiclass problem (Figure 3.5 **(b)**) NPs moves x towards the nearest decision boundary, preferring the class C,

since the perturbation required to move x from the class A to the class C is less than that required to move to the B.

While Figure 3.5 **(a)** represents the cases of ResNet24 on CIFAR-10, and MobileNet-v3 on FCGV7, Figure 3.5 **(a)** represents the case of ResNet50 on Dogs vs. Cats. Therefore, since ResNet50 is quite sure that an input image is either a cat or a dog – the likelihood that such an image belongs to the correct class is almost 1 –, and the problem such a network address is a binary classification one, a greater perturbation is required by NPs to craft a successful adversarial image. Therefore, by relaxing the *similarity constraint* (Figure 3.4 **(b)**), the success-rate of NPs while targeting ResNet50 almost doubles.

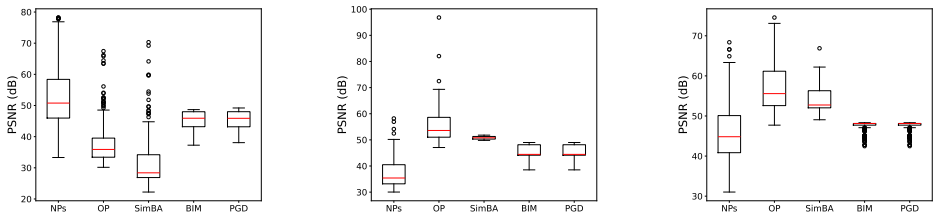
Comparison with SotA and Discussion

Finally, we compared results of NPs with iterative adversarial attacks from the SotA. We compared our attack with Basic Iterative Method (BIM) [20], Projected Gradient Descent (PGD) [22], OP [25] and Simple Black-Box Adversarial Attack (SimBA) [94] in terms of both the success-rate and PSNR, i.e., ow well an attack succeeds in creating an adversarial image that is as more similar as possible to the original image. Note that, in such comparison, we make NPs undergoes the similarity constraint of $\Omega_w = 70$ (the minimal PSNR was 30 dB).

	NPs	**OP** [25]	**SimBA** [94]	**BIM** [20]	**PGD** [22]
ResNet24	**91.9**	97.2	32.6	99.6	99.7
ResNet50	**22.01**	16	0.4	97.8	97.8
MobilNet-v3	**89.17**	26.83	3.67	100	100

Table 3.4: Success-rate of NPs, OP, SimBA, BIM, and PGD against ResNet 24, Resnet50, and Mobilenet-v3.

As Table 3.4 depicts, the success-rate of NPs is greater than the ones of black-box attacks, and lower than the ones of white-box attacks, for all the



(a) ResNet24. **(b)** ResNet50. **(c)** MobileNet-v3.

Figure 3.6: Distribution of the PSNRs while targeting ResNet24 **(a)**, ResNet50 **(b)**, and MobileNet-v3 **(c)** with BIM, PGD, NPs, OP, and SimBA.

considered CNNs. Moreover, as Figure 3.6 depicts, the PSNR is distributed worse while targeting the CNNs with NPs, rather than while leveraging other adversarial methodologies from the SotA. NPs performs worse than white-box attacks in both the success-rate and the PSNR. However, by lowering the PSNR – note that, with the *similarity constraint*, we imposed the minimum PSNR equal to 30 dB –, we ensured that the success-rate of NPs is much greater than the ones of other black-box attacks, despite the perturbations are still not visible to human eye.

3.1.3 Discussion

The experimental results of previous Section confirm what have emerged from the scientific-literature, namely that gray-box attacks are a trade-off between white-box attacks and black-box ones. Finally, all the black-box attacks – including NPs – performs much worse than white-box attacks while targeting binary-classification problems. However, by further relaxing the *similarity constraint*, the success-rate of NPs could increase even more. In addition, the optimal functioning of NPs is contingent upon the utilization of the ReLU or its variants as activation function within the target CNN.

3.2 Incremental Pixels: a Black Box Adversarial Attack

IPs is an iterative black-box score-based adversarial attack, based on the OP one [25], that contribute to the SotA by facing the following challenges. First, due to its black-box nature, it require access only of the inputs and outputs of the target CNNs – more specifically to the outputs of the softmax layer, since it is score-based –, in response to the Challenge 1. Furthermore, the attack leverages deterministic approaches for computing and applying the perturbation to the pixels, by relying on bisection methodologies, in response to the Challenge 5. Finally, the attack perturb the pixels of the target image by leveraging the statistics of its local neighborhood. This approach, in conjunction with deterministic methods for computing the perturbation, facilitates enhancing the imperceptibility, in response to the Challenge 4. In the following, we will detail how IPs addresses the mentioned challenges, and we will present experimental results that prove the suitability of the attack.

3.2.1 Methodology

IPs is a score-based attack that crafts an adversarial image by modifying an increasing number of pixels, by applying to them an increasing perturbation, carefully controlled with bisection methodologies. Such an attach is an improvement of OP [25], since we introduced the following contributions:

- The number of pixels that IPs modifies is not fixed a priori, but increased until a maximum number, starting from a single pixel, with bisection methodologies;

- The perturbation is carefully selected with bisection methodologies, until a successful adversarial image is not crafted, or until a maximum value for the perturbation is reached.

In the following, we will first provide preliminaries in order to comprehend the methodologies we proposed. Subsequently, we will analyze in detail each step of IPs.

Preliminaries

Prior to an in-depth exploration of IPs, the reader is provided with preliminaries and definitions in order to better understand our contribution. For this reason, we revised some definitions given in Sections 2.1.1 and 2.2.1. Here, we define a test image x as a set of P pixels $(p_{i.j})$, such that: $x = \{p_{i,j}\} : i \in [0, h-1], j \in [0, w-1]$. Each pixel $p_{i,j}$ is composed of c values, depending on the number of the c channels. For instance, each pixel of an RGB image is a vector of 3 channels, e.g., $p_{i,j} = [R_{i,j}^{value}, G_{i,j}^{value}, B_{i,j}^{value}]$. Furthermore, we revised the definition (2.6). Indeed, here we define a perturbed sample as in Equation (3.4).

$$\hat{x} = x + \Delta \tag{3.4}$$

The reason is that the perturbation Δ alters more than one pixel of the target image x. Indeed, Δ is defined as in Equation (3.5).

$$\Delta = \mathbf{M} \odot \delta \tag{3.5}$$

Indeed, in the Equation 3.5, $\mathbf{M}$ is a mask of P elements $m_{i,j}$, such that $m_{i,j} \in \{0,1\}\ \forall i \in [0, h-1], j \in [0, w-1]$. Consequently, if $m_{i,j} = 1$, the pixel $p_{i,j}$ will be altered; otherwise, the perturbation will not be applied to it. δ is a matrix of P perturbations $\delta_{i,j}$, and each element $\delta_{i,j}$ express the perturbation to apply to the pixel $p_{i,j}$. Clearly, each perturbation $\delta_{i,j}$ is composed of c channels, as for the pixel $p_{i,j}$. Finally, $\odot$ is the element-wise multiplications between the mask M and the perturbation-matrix δ. Therefore,

the perturbation can be expressed as:

$$\Delta = \begin{bmatrix} m_{0,0} \cdot \delta_{0,0} & \cdots & m_{0,w-1} \cdot \delta_{0,w-1} \\ \vdots & \ddots & \vdots \\ m_{h-1,0} \cdot \delta_{h-1,0} & \cdots & m_{h-1,w-1} \cdot \delta_{h-1,w-1} \end{bmatrix} \tag{3.6}$$

It follows that a generic $\Delta_{i,j} = m_{i,j} \cdot \delta_{i,j}$ is equal to $\delta_{i,j}$ if $m_{i,j} = 1$, meaning that the perturbation $\delta_{i,j}$ will be applied to the pixel $p_{i,j}$. Otherwise, $\Delta_{i,j} = 0$, and the pixel $p_{i,j}$ will not be perturbed.

In light of the preceding discussion, the perturbed image $\hat{x}$ can be expressed as follows:

$$\hat{x} = \begin{bmatrix} p_{0,0} & \cdots & p_{0,\beta} + \delta_{0,\beta} & \cdots & p_{0,w-1} \\ \vdots & & \ddots & & \vdots \\ \cdots & p_{\alpha,\gamma} + \delta_{\alpha,\gamma} & \cdots & \cdots & p_{\alpha,w-1} \\ \vdots & & \ddots & & \vdots \\ p_{h-1,0} & \cdots & \cdots & \cdots & p_{h-1,w-1} \end{bmatrix} \tag{3.7}$$

Clearly, Equation (3.7) assumes that the mask $\mathbf{M}$ of is composed with $P-2$ elements equal to 0, except for the elements $m_{0,\beta}$ and $m_{\alpha,\gamma}$ that are equal to 1, indicating that only the pixels $p_{0,\beta}$ and $p_{\alpha,\gamma}$ of the image x will be altered with the perturbations $\delta_{0,\beta}$ and $\delta_{\alpha,\gamma}$, respectively.

Incremental Pixels Algorithm

Here, the IPs algorithm will be explained in each step. (i) First, we will detail the number of pixels to perturb within the input image, followed by (ii) the selection of the pixels to perturb once the number is established. Finally, (iii) the computation of the perturbation to apply to each pixel will be explained.

Assume that IPs selects a single pixel $p_{\alpha,\beta}$ to modify. It follows that $\Delta_{i,j} = 0 \, \forall i \in [0, h-1] \, , i \neq \alpha \, and \, \forall j \in [0, w-i] \, , w \neq \beta$, and $\Delta_{\alpha,\beta} = \delta_{\alpha,\beta}$. Note that

the value of the perturbation $\delta_{\alpha,\beta}$ is constrained between a minimum and a maximum value, ω and Ω, respectively, meaning that $\omega \leq \delta_{\alpha,\beta} \leq \Omega$. Clearly, the values ω and Ω depend on both the original value of $p_{\alpha,\beta}$ and on the minimum and maximum values allowed by the context, γ and Γ, respectively. For instance, in case of RGB images whose pixels are in the range $[0,255]$, $\gamma = 0$ and $\Gamma = 255$. Consequently, $\omega = \gamma - p_{\alpha,\beta}$, and $\Omega = \Gamma - p_{\alpha,\beta}$. Finally:

$$\gamma - p_{\alpha,\beta} \leq \delta_{\alpha,\beta} \leq \Gamma - p_{\alpha,\beta} \tag{3.8}$$

Algorithm 4 Procedure for selecting the perturbation, and the number of pixels to perturb, in order to craft an adversarial image.

procedure SELECTPERTURBATION(x, y, ω, Ω, μ, $maxP$, $minP$, ψ, σ, r, V)
  $x^{adv} \leftarrow x$
  while $\Omega - \omega > \mu$ **do**
    $\delta \leftarrow \frac{\Omega+\omega}{2}$
    $adversarialFound \leftarrow$ False
    while $maxP - minP > 1$ **do**
      $Ps \leftarrow \left\lfloor \frac{maxP+minP}{2} \right\rfloor$
      $\Theta \leftarrow$ ipDifferentialEvolution$(x, y, Ps, \delta, \psi, \sigma, r, V)$
      for $x' \in \Theta$ **do**
        $y_a \leftarrow \text{argmax}(f(x'))$
        if $y_a \neq y$ **then**
          $x^{adv} \leftarrow x'$
          $adversarialFound \leftarrow$ True
          $maxP \leftarrow Ps$
        else
          $minP \leftarrow Ps + 1$
        end if
      end for
    end while
    if $adversarialFound = True$ **then**
      $\Omega \leftarrow \delta$
    else
      $\omega \leftarrow \delta + 1$
    end if
  end while
  return x^{adv}
end procedure

Procedure 4 depicts how both the number of pixels, and the perturbation to apply to them, are selected. In specific, it is a two-level binary search algorithm which aims to identify the minimal perturbation required to cause the target CNN to misclassify the input test-image x while modifying as few pixels as possible. In addition to the aforementioned x, the procedure also needs (i) the correct class y associated to the image, (ii) the initial range $[minP, maxP]$ in order to compute the pixels to be perturbed, that is, the minimum and the maximum number of pixels to perturb, (iii) the range $[\omega, \Omega]$ for the perturbation δ, that is, the minimum and the maximum value for the perturbation, respectively, as well as (iv) the convergence threshold μ for the perturbation, needed to stop the binary search for the perturbation. (v) In addition, since the proposed attack leverage a genetic algorithm in order to craft the adversarial image, the procedure requires a number of generation ψ, and the dimension of the population σ. (vi) Finally, r and V are utilized to determine the sign of the perturbation for each pixel, as we will explain better later.

Fixed a perturbation δ, the inner loop (lines 6 to 19) computes the number Ps, that is, the number of pixels that will be perturbed (line 7) in the range $[minP, maxP]$. Consequently, the procedure PerturbImage leverages a genetic algorithm to generate a set of perturbed images Θ, crafted by applying the fixed δ to Ps pixels. Please note that the same perturbation δ will be applied to all the selected Ps pixels, while the sign (whereas the perturbation will be added or subtracted) will be decided by the procedure ipDifferentialEvolution for each specific pixel. Finally, the search bound for the number of pixels is updated (lines 11 to 17). In details: (i) if the population POP contains a successful adversarial image, the upper bound is updated by assigning $maxP = Ps$. This is done in order to identify a better adversarial image, whose number of perturbed pixels is less than that of the current one. (ii) Otherwise, the number of pixels to be perturbed is increased with the objective of crafting a successful adversarial image with such a fixed perturbation.

Similarly, the outer loop computes the specific perturbation δ within the range $[\omega, \Omega]$, using a binary search algorithm. The upper and lower bounds are updated in accordance with the outcome of the perturbation, either indicating the successful crafting of an adversarial image or the absence thereof, respectively. The binary searches allow addressing both the Challenge 5 and the Challenge 4.

Algorithm 5 Incremental Pixel implementations of the Differential Evolution genetic algorithm for computing a population of perturbed images, given a fixed perturbation a number of pixels to perturb.

procedure IPDIFFERENTIALEVOLUTION(x, y, Ps, δ, ψ, σ, r, V)
  $\Theta \leftarrow \emptyset$
  $POP \leftarrow \text{initPopulation}(x, Ps, \sigma)$
  for $j \leftarrow 0$; $j < \psi$; $j \leftarrow j+1$ **do**
    $POP' \leftarrow \emptyset$
    for $i \leftarrow 0$; $i < \sigma$; $i \leftarrow i+1$ **do**
      repeat
        $\mathcal{P}_a, \mathcal{P}_b, \mathcal{P}_c \leftarrow \text{randomPick}(POP)$
        $\mathcal{P}_{new} \leftarrow \mathcal{P}_a + H \cdot (\mathcal{P}_b - \mathcal{P}_c)$
        for $\mathcal{P} \in \{\mathcal{P}_{new}, \mathcal{P}_a, \mathcal{P}_b, \mathcal{P}_c\}$ **do**
          $x' \leftarrow \text{perturbImage}(x, \mathcal{P}, \delta, r, V)$
        end for
      until $||x - x_{new}||_\infty < \delta$
      $\rho_{\{new\}}, \rho_{\{a\}}, \rho_{\{b\}}, \rho_{\{c\}} \leftarrow f(x')$, $x' \in \{x_{new}, x_a, x_b, x_c\}$
      $\xi \leftarrow \mathcal{P}$, $\mathcal{P} : \min\{\rho', \ \rho : \rho' \in \{\rho_{\{new\}}, \rho_{\{a\}}, \rho_{\{b\}}, \rho_{\{c\}}\}$
      $POP' \leftarrow POP' \cup \{\xi\}$
    end for
    $POP \leftarrow POP'$
  end for
  for $\mathcal{P} \in POP$ **do**
    $\hat{x} \leftarrow \text{perturbImage}(x, \mathcal{P}, \delta, r, V)$
    $\Theta \leftarrow \Theta \cup \{x\}$
  end for
  return Θ
end procedure

The genetic procedure utilized by the IPs attack to generate an adversarial image is illustrated in Procedure 5. Specifically, it implements the Differential Evolution (DE) genetic algorithm [95]. The initial population, denoted by

POP, is generated in the first step of the procedure. This population consists of σ candidate solutions, as outlined in line 3. In fact, each solution is a set of *Ps* pixels, whose coordinates are selected by using a uniform distribution within the ranges $[0, h-1]$, $[0, w-1]$ (length and width) of the target image x. Subsequently, a new child solution, designated as $\mathcal{P}_{new}$, is generated by relying on the mutation operator (line 9). This operator randomly selects three distinct candidate solutions $\mathcal{P}_a$, $\mathcal{P}_b$, and $\mathcal{P}_c$ from the initial population, and generates the child solution in a manner analogous to that of the OP attack, as depicted in Equation (2.13). It should be noted that, as with the OP attack, the constant H is set to 0.5. For each solution, the procedure generates a perturbed image by applying the previously fixed perturbation δ to all pixels of the candidate solution. This results in four distinct perturbed images x_{new}, x_a, x_b, and x_c. However, as the primary objective of IPs is to generate an adversarial image with minimal perturbation, we further enhance the DE mutation operator by imposing an L_∞-distance constraint, in order to face the Challenge 4. A new child solution $\mathcal{P}_{new}$ is generated until:

$$||x - x_{new}||_\infty \leq \delta \tag{3.9}$$

This means that, once generated the perturbed image x_{new} by applying the perturbation δ to all the pixel of the $\mathcal{P}_{new}$ child-solution, such solution is accepted to be a candidate solution only if the maximum distance between the pixels of the original image x and x_{new} is δ. The surviving solution among $\mathcal{P}_{new}$, $\mathcal{P}_a$, $\mathcal{P}_b$, and $\mathcal{P}_c$, i.e., the new solution of the new population POP', will be the one related to the perturbed image that minimize the ρ^*, that is, the likelihood that such a perturbed image belongs to the original class y, once classified by the target model (lines 14 to 16).

Once generated the population, for each solution (set of *Ps* pixels) the Procedure 5 generates the related perturbed images. Such images are then evaluated by Procedure 3 in order to either find an adversarial image, or

modify the number of pixels to modify and the associated perturbation, as explained before.

Algorithm 6 Procedure for perturbing the input image x, once fixed the number of pixel to perturb, and the perturbation to apply to them.

```
1:  procedure PERTURBIMAGE(x, 𝒫, δ, r, V)
2:      x̂ ← x
3:      for p ∈ 𝒫 do
4:          η ← computeMeanOfNeighborhood(x, p, r)
5:          if η ≥ V then
6:              x̂ ← x̂[x_p + δ]
7:          else
8:              x̂ ← x̂[x_p − δ]
9:          end if
10:     end for
11:     return x̂
12: end procedure
```

Finally, Procedure 6 depicts how the image x is actually perturbed. Once Procedure 3 fix the perturbation δ and the number of pixels to perturb, and once Procedure 5 generates a candidate solution $\mathcal{P}$ (set of pixels), the Procedure 6 decides the sign of the perturbation δ, hence whereas such a perturbation should be additive or subtractive. The sign of the perturbation is chosen basing on the neighborhood of the specific pixel to which the perturbation has to be applied. In specific, given a pixel $p \in \mathcal{P}$, Procedure 6 computes the mean value η of the pixels in a neighborhood of radius r of p (line 4). If $\eta \geq V - V$ is a configurable threshold –, the perturbation δ will be added to the pixel p; otherwise, it will be subtracted. Such a reasoning allow to change as less as possible the behavior of groups of pixels, while introducing a perturbation.

Differences with One Pixel

The IPs attack is an improved version of the OP attack proposed by Su *et al.* in [25]. It is thus deemed pertinent to delineate the distinctions between the

two attacks and to highlight the enhancements introduced by the proposed attack.

First, OP modifies either a single pixel – which could be not enough to craft successful adversarial images –, or a number of pixels fixed a priori. Conversely, IPs computes the number of pixels to perturb with bisection methodologies, in order to (i) perturb as fewer pixels as possible (IPs carefully computes the number of pixels to perturb), and (ii) generate an adversarial image with few queries (the number of pixels to perturb is computed with bisection methodologies, rather than with incremental ones).

Second, IPs carefully select a perturbation to apply to the pixels, with bisection methodologies. In particular, OP lets the DE genetic algorithm to select both the coordinates of pixels to modify and the perturbation to apply, by following the Uniform and the Gaussian distribution, respectively. Indeed, each DE candidate solution is a quintuple holding the (i, j) coordinates of the pixel to modify, and the RGB values to apply to such pixels. Conversely, IPs lets DE to select only the (i, j) coordinates of the pixel to perturb, while the perturbation to apply is computed as previoussly depicted. Meaning that, the magnitude of the perturbation is computed with bisection methodologies, aiming to minimize that, while the sign of the perturbation is decided basing on the neighborhood of such pixel. Finally, IPs is able to target any Deep Neural Network (DNN), despite the dataset, while OP is specifically designed to target exclusively the CIFAR-10 [156] and the ImageNet [55] datasets.

3.2.2 Experimental Evaluation

In order to evaluate the IPs attack, we targeted different DNNs and datasets from the scientific literature. The following Sections provide the reader with details related to the experimental setup, and the analysis of the experimental results. Finally, we compare our attack with other SotA attacks.

Experimental Setup

In this Subsection, we provide the reader with details related to the DNNs and datasets we exploit to evaluate our attacks, and how we configured IPs attacks in both the first and the second stage.

Networks and Datasets We consider several datasets and different DNNs. In particular, we targeted three networks from the ResNet family [155], i.e., ResNet-50D and ResNet-50C, and ResNet-24 trained to cope with the Dogs vs. Cats [157], the CIFAR-100 [156], and the CIFAR-10 [156] datasets, respectively. Finally, we targeted MobileNet-v3 [158] and VGG-16 [160], pre-trained on the ImageNet [55] dataset, and fine-tuned on the FGCV7 dataset [159] and Fruits-360 [161] datasets, respectively.

Both the CIFAR-10 and CIFAR-100 datasets consist of 60 thousand 32×32 RGB images, each belonging to 10 and 100 classes, respectively. The Dogs vs. Cats datasets consist of 25 thousand 224×224 RGB images belonging to the classes *cat* and *dog*, while the FCGV7 dataset is composed of 3650 224×224 high-quality RGB images of multiple real-life symptoms of diseases affecting plants. Finally, the Fruits-360 dataset consists of 94110 100×100 RGB images belonging to 141 different classes, i.e., fruits and vegetables. Please note that we rescaled the images of the Fruits-360 from 100×100 to 224×224, in order to use the VGG-16 network.

The datasets were partitioned in order to allocate 60% of the images for training purposes, 20% to validate the training process, and reserve the remaining 20% for use as test dataset. Finally, for the following experiment, we consider 360 images belonging to 36 different classes of the Fruit-360 test dataset, and, we use the 35% of the images from the test dataset that are correctly classified by the target model, for each of the above DNNs and datasets.

Table 3.5: Accuracy of ResNet-24, ResNet-50C, ResNet-50D, MobileNet-v3, and VGG-16 on the CIFAR-10, CIFAR-100, Dogs vs. Cats, FGCV7, and Fruits-360 test datasets, respectively.

DNN	Dataset	Accuracy (%)
ResNet-24	CIFAR-10	84.38
ResNet-50C	CIFAR-100	80.31
ResNet-50D	Dogs vs. Cats	97.01
MobileNet-v3	FGCV7	98.08
VGG-16	Fruits-360	84.67

Table 3.5 reports the accuracy of each DNN on the respective test dataset, and the number of images considered for the experiment (the 35% of each test dataset).

Incremental Pixels Setup In order to target the above-mentioned DNNs, we set the parameters of IPs in a specific manner depending on the size of the images belonging to the test set. We refer to the parameter of Procedure 3 as parameter of the proposed attack.

Since we target RGB images, each pixel is represented as a vector of three channels $[R,G,B]$, whose each value is in the range $[0,255]$. Thus, we set the minimum perturbation $\omega = 1$, while the maximum one $\Omega = 100$, hence the perturbation δ is computed between the range $[1,100]$. In addition, we set the bisection threshold $\mu = 1$, in order to try to apply also the minimum perturbation to the image. For what pertains the number of pixel, we set $minP = 1$, meaning that at least one pixel has to be perturbed. Conversely, we set $maxP$ to be the 1% of the total pixels of the target image. Consequently, $maxP = 10$ in case of 32×32 images (both CIFAR-10 and CIFAR-100 datasets) – meaning that it modifies to the most $10 \times 3 = 30$ different channels – , and $maxP = 501$ – at most 1503 different channels –while considering 224×224 images (FCGV7, Dogs vs. Cats, and Fruit-360 datasets).

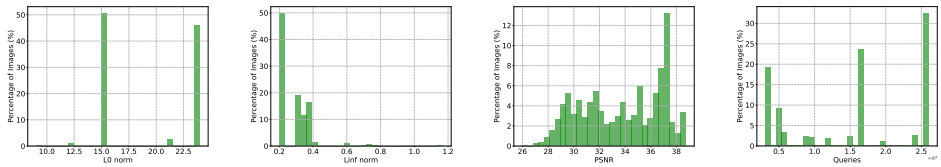
Furthermore, IPs leverages the DE genetic algorithm to find the pixels to perturb. We let DE to evolve within 20 generations ($\psi = 20$), and a population-size of 5 ($\sigma = 5$) while targeting 224×224. Conversely, DE evolves within 10 generations, and a population size of 5, while targeting 32×32 images.

Finally, as Procedure 6 depicts, once selected a perturbation δ and a pixel p to perturb, IPs employs a neighborhood radius of r to ascertain whether the perturbation should be additive or subtractive, based on the average value of pixels within the designated area. In the event that the aforementioned average exceeds a specific threshold value, designated as V, the perturbation δ is additive. Otherwise, the perturbation is subtractive. We set r equal to 2 while targeting 32×32 images, and $r = 3$ while targeting both and 224×224 images. This means that IPs examines a number of pixels that is less equal to 24 and 48, respectively, depending on whereas the pixel p to perturb is situated near the edge of the image or not. Finally, the threshold value V was set to $\lfloor \frac{255}{2} \rfloor = 127$. Please note that the perturbation δ is applied to all the 3 $[R,G,B]$ channels of a pixel, meaning that the sign of the perturbation is decided for each channel independently.

Table 3.6: Success-rate of IPs against ResNet-24, ResNet-50C, ResNet-50D, MobileNet-v3, and VGG-16 on their respective datasets.

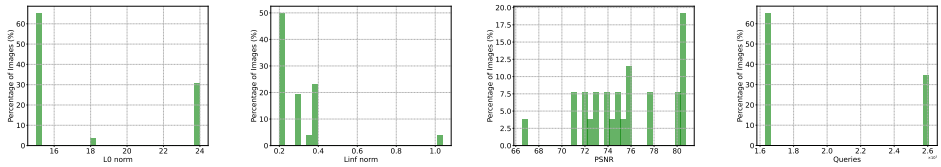
DNN	**Dataset**	**IPs Success Rate (%)**
ResNet-24	CIFAR-10	49.75
ResNet-50C	CIFAR-100	0.92
ResNet-50D	Dogs vs. Cats	21.18
MobileNet-v3	FGCV7	16.8
VGG-16	Fruits-360	18.87

Result Analysis Table 3.6 depicts the success rate of IPs against the DNNs we considering, once setting to 1% the maximum number of pixels, within an input test-image, to perturb, and to 100 the maximum perturbation to be applied. In particular, IPs has a high success rate – almost 50% – against



(a) L_0 Norm **(b)** L_∞ Norm **(c)** PSNR **(d)** Queries

Figure 3.7: Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPs while targeting ResNet-24 on the CIFAR-10 test dataset.

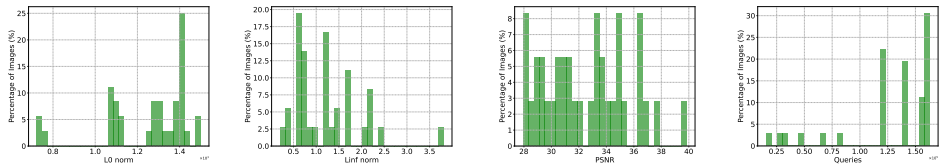


(a) L_0 Norm **(b)** L_∞ Norm **(c)** PSNR **(d)** Queries

Figure 3.8: Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPs while targeting ResNet-50C on the CIFAR-100 test dataset.

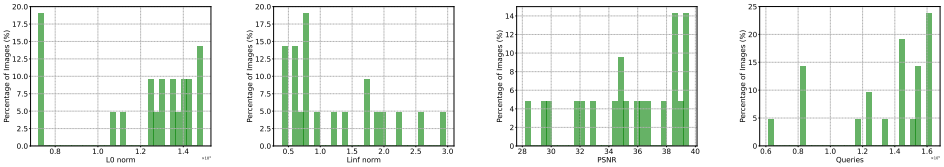
ResNet24 on the CIFAR-10 datasets, while the success rate is approximately 20% while targeting ResNet-50D, MobileNet-v3, and VGG-16 on the Dogs vs. Cats, FGCV7, and Fruits-360, respectively. However, the success-rate is low – almost 1% while targeting ResNet-50C on the CIFAR-100 dataset. Such results could depend on different factors.

1. The number of classes of the target dataset, i.e., the smaller, the higher could be the success rate of IPs. Indeed, CIFAR-100 images belong to 100 different classes. Therefore, is more difficult to lead ResNet-50C to misclassify with a controlled and smaller perturbation. Note that such an issue was discussed in [24, 162]. Indeed, authors claimed that the more difficult is the classification-task, the more difficult is to craft successful adversarial images.



(a) L_0 Norm **(b)** L_∞ Norm **(c)** PSNR **(d)** Queries

Figure 3.9: Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPs while targeting ResNet-50D on the Dogs vs. Cats test dataset.

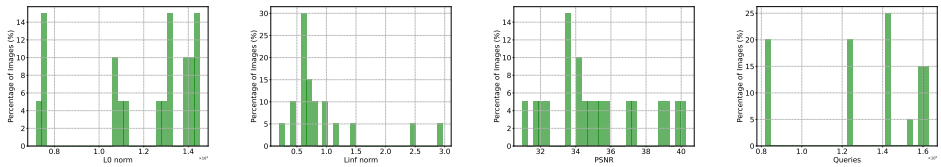


(a) L_0 Norm **(b)** L_∞ Norm **(c)** PSNR **(d)** Queries

Figure 3.10: Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPswhile targeting MobileNet on the FGCV7 test dataset.

2. The number of layers of the target DNN. Indeed, the success rate against ResNet-24 is the highest, and the number of layers of ResNet-24 is the smallest. Conversely, deeper DNNs have a stronger feature-aggregation capability, that leads to a stronger robustness against small and localized perturbation [23, 163].

3. The confidence-score associated to the predicted class. Indeed, the closer such a score is to 1 – the network is *almost sure* that the image being to the predicted class –, the more difficult is to craft the adversarial image, in terms of magnitude of the perturbation applied [19]. Indeed, the confidence scores of predictions of ResNet-50C are almost always around 1.



(a) L_0 Norm **(b)** L_∞ Norm **(c)** PSNR **(d)** Queries

Figure 3.11: Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPs while targeting VGG-16 on the Fruit-360 test dataset.

Therefore, it is clear that in order to increase the success-rate of IPs against ResNet-50, it is necessary to adjust the parameters of the attack. Indeed, the maximum perturbation Ω, the maximum number of pixels *maxP*, the generations and the population-size (ψ and σ, respectively), and the radius r should undergo a proportional increase.

Figure 3.7 depicts how the IPs algorithm behaves in order to craft adversarial samples against ResNet-24. In particular, IPs generally modifies either 15 or 24 channels, 5 and 8 pixels, respectively, in order to craft successful adversarial images, while maintaining the perturbed images almost indistinguishable from the original one (the L_∞ norm is typically distributed between 0.2 and 0.4, and the PSNR is in the range $[28, 38]$, resulting in a good quality of the adversarial images). Finally, the number of queries required to craft the adversarial images is generally either 400, 1600, or 2500. For what pertains to ResNet-50C, Figure 3.8 depicts results of the attack. The number of channels modified by IPs, in order to craft adversarial images, is still either 15 or 24, as for ResNet-24. Similarly, both the number of queries required to craft images, and the L_∞ norm of the resulting images, are in the same range of the case of ResNet-24. Conversely, the PSNR is distributed in the range $[67, 80]$, meaning that the adversarial image are highly indistinguishable from the original ones.

For what pertains to ResNet-50D, Figure 3.9 depicts the results while IPs targets such a network on the Dogs vs. Cats dataset. Clearly, the number

of channels that IPs modifies is greater w.r.t. the other dataset, since the datasets in question is composed of images that are 49 times bigger than the ones of the other datasets. Similarly, the L_∞ norm is distributed in the range $[0.5, 2.5]$, except for some outliers, and the number of queries required to fool ResNet-50D is two orders of magnitude larger than that required to fool both ResNet-24 and ResNet-50C. Finally, similarly to ResNet-24 on the CIFAR-10 dataset, the PSNR of the Dogs vs. Cats adversarial images is distributed in the range $[28, 40]$, resulting in a good-quality adversarial images.

Finally, Figures 3.10 and 3.11 depicts results related to MobileNet on FGCV7, and VGG-16 on Fruits-360, respectively. For both the DNNs, the L_0 norm is distributed in the range $[0.8, 1.4] \times 10^3$, and the L_∞ in the range $[0.5, 3]$. The PSNR is in the range $[28, 40]$, and $[32, 40]$ for FGCV7 and Fruit-360 adversarial images, respectively, meaning that the adversarial images are indistinguishable from the original ones. Finally, the number of queries required to fool MobileNet and VGG-16 are $[0.6, 1.6] \times 10^5$, and $[0.8, 1.6] \times 10^5$, respectively.

By observing the results above described, it is possible to state that IPs is able to craft adversarial images that are almost indistinguishable from the original ones, for all the considered DNNs on the respective datasets. Furthermore, both the number of channels, hence the number of pixels, and the number of queries to fool a DNN, increases as the size of the target image increases.

Comparison with SotA

In this Section, we compare the proposed IPs with OP – note that IPs is is an improvement of the OP attack – with other two attacks from the SotA, that is, Pixle (PX) [164] and Square Attack (SQ) [26]. All the results are in reported in Table 3.7. In specific, the 1^{st}, the 2^{nd} and the 3^{th} columns of Table 3.7 show the target DNN, the corresponding dataset, and the attack under analysis, respectively. Subsequently, the 4^{th} column of Table 3.7 depicts

the success-rate of each attack. Finally, columns from the 5^{th} to the 8^{th} of Table 3.7 depict the L_0 norm, the L_∞ norm, the PSNR, and the number of queries for each attack, respectively. However, for the sake of clarity, we discuss the separately the results in the two following Sections. In specific we compare and discuss results. (i) between IPs and OP in 3.2.2, and (ii) between IPs and both SQ and PX in 3.2.2.

Table 3.7: Results obtained while attacking ResNet-24, ResNet-50C, ResNet-50D, MobileNet-v3, and VGG-16 on their respective datasets, with the proposed attack IPs, OP while targeting a single pixel (OP1), OP while targeting 1% of the pixels (OP1%), SQ, and PX. For each score, we show the mean and the variance, if present, calculated over all the images on that test dataset.

DNN	Dataset	Attack	Success Rate	L_0 Norm	L_∞ Norm	PSNR	Queries
ResNet-24	CIFAR-10	IP	49.75	19.26 ± 4.48	0.28 ± 0.1	33.72 ± 3.12	1535.13 ± 914.56
		OP1	11.18	3.0 ± 0	0.38 ± 0.03	39.88 ± 2.0	193.07 ± 67.23
		OP1%	44.90	29.85 ± 0.71	0.4 ± 0.06	31.42 ± 1.31	1590.0 ± 710.5
		SQ	100	2434.27 ± 208.02	0.17 ± 0.13	28.7 ± 6.27	9.9 ± 4.31
		PX	62.72	118.2 ± 22.71	0.71 ± 0.16	24.03 ± 2.71	101.0 ± 0.0
ResNet-50C	CIFAR-100	IP	0.92	17.88 ± 4.2	0.3 ± 0.17	75.59 ± 3.83	1968.54 ± 474.49
		OP1	0.28	3.0 ± 0	0.34 ± 0.1	81.91 ± 5.34	245.0 ± 0.0
		OP1%	0.78	29.86 ± 0.64	0.41 ± 0.08	71.74 ± 2.7	2405.0 ± 0.0
		SQ	100	2363.53 ± 65.42	131.72 ± 16.91	10.2 ± 0.0	1.08 ± 0.55
		PX	70.64	111.87 ± 27.2	182.08 ± 41.36	19.13 ± 2.87	101.0 ± 0.0
ResNet-50D	Dogs vs. Cats	IP	21.18	1250.0 ± 206.26	1.23 ± 0.75	32.47 ± 3.09	128691.33 ± 41617.51
		OP1	0.0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
		OP1%	12.35	1494.05 ± 6.15	0.67 ± 0.17	33.04 ± 0.97	102112.67 ± 31627.19
		SQ	81.18	121520.54 ± 8426.48	0.63 ± 0.27	17.69 ± 3.65	12.25 ± 4.68
		PX	28.82	5425.86 ± 1468.37	0.89 ± 0.11	23.23 ± 2.44	101.0 ± 0.0
MobileNet-v3	FGCV7	IP	16.8	1222.86 ± 268.02	1.22 ± 0.77	35.42 ± 3.55	133307.0 ± 31378.46
		OP1	1.6	3.0 ± 0	0.39 ± 0	55.63 ± 0.99	163.0 ± 115.97
		OP1%	15.6	1494.06 ± 4.85	0.71 ± 0.13	32.89 ± 0.55	97581.81 ± 30910.9
		SQ	100	118082.1 ± 8876.03	0.3 ± 0.21	25.16 ± 5.22	12.04 ± 4.25
		PX	12	5920.13 ± 1403.29	0.82 ± 0.09	25.49 ± 2.18	101.0 ± 0.0
VGG-16	Fruits-360	IP	18.87	1189.5 ± 260.37	0.92 ± 0.68	35.25 ± 2.77	132339.7 ± 29231.6
		OP1	0.0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
		OP1%	13.21	1492.93 ± 4.98	0.64 ± 0.15	33.36 ± 0.63	120245.0 ± 0.0
		SQ	100	121763.69 ± 8488.89	0.75 ± 0.28	16.77 ± 5.74	13.73 ± 3.63
		PX	29.24	6350.5 ± 1256.4	0.95 ± 0.06	21.9 ± 2.12	101.0 ± 0.0

Comparison with One-Pixel Since IPs is an improvement of the OP attack, in this Paragraph we compare the proposed attack with OP. In specific, we configured the OP consistently with the configuration of IPs. Therefore: (i) we configure OP in order to target both a single pixel and a number of pixels that is the 1% of the overall number of pixels of the target image – 1 and 10 pixels in case of 32×32 images, and 1 and 501 pixels for 224×224

images –. (ii) OP lets the DE algorithm to evolve within 20 generation and a population-size of 5 in case of 224×224 images, and within 10 generation and a population-size of 5 in case of 32×32 images.

The 4^{th} column of Table 3.7 depicts the success-rate of IPs, and OP while targeting both a single pixel of the target image, and 1% pixels. Henceforth, we will refer to the 1 pixel and the 1% pixels configurations of OP as OP1 and OP1%, respectively. The success-rate of OP is always lower than IPs. In specific, the success rate of OP1 is much lower than IPs, and it is 0.0% in case of the Dogs vs. Cats and Fruit-360 datasets, respectively. Moreover, the success-rate of OP1% is slightly lower, despite IPs targets a number of pixels that is maximum 1%, meaning that IPs could perturb fewer pixel than OP1%, and its success-rate is even higher. The reason is that, despite the number of pixels perturbed by OP could be greater than the pixels perturbed by IPs, the perturbation is selected randomly, while IPs selects the perturbation carefully. Therefore, once selected a pixel to perturb, the perturbation applied to it by OP could be too small to fool the target DNN. Such reasoning applies for both OP1 and OP1%.

Subsequently, in all the cases – except for the ResNet-50D and the VGG-16 cases, where the success-rate of OP1 is 0% – the L_0 norm, the L_∞ one, and the number of queries of OP1 are smaller than the ones of IPs, while the PSNR is higher. The reason is that OP1 modifies a single pixel (3 channels) of the target images, resulting in an L_0 always equal to 3 (3 channels modified), and, since the perturbation is introduced to a single pixel, the resulting adversarial image is more similar to the original image than the adversarial image generated by IPs (the PSNR is higher, and the L_∞ is lower). Finally, the number of queries that OP1 requires to fool the DNNs is significantly lower than the one required by IPs. Therefore, OP1 generates adversarial images with few queries, and by introducing the smallest perturbation ever to the original image. However, the success-rate of OP1 is drastically low w.r.t. IPs, as previously discussed. Conversely, the L_0 norm of the adversarial

images generated by OP1% is higher than the IPs case, meaning that OP1% perturbs more pixels (more channels) than IPs. Furthermore, the PSNR of adversarial images generated by OP1% is generally lower than the IPs case – except while targeting ResNet-50D, where the average PSNR is higher, as well as the variance –, meaning that IPs-adversarial images are more similar to the original ones than the OP1%-adversarial images. The same conclusion applies for the L_∞ norm. However, IPs require more queries in to generate adversarial images, while targeting 224×224 images than OP1%. Otherwise, the number of queries is smaller while targeting 32×32 images.

Therefore, as demonstrated by the previous experimental results, IPs improves the OP attack and performs well than the latter. Indeed, the success-rate of IPs is always higher than OP and, in comparable settings (IPs and OP1%), IPs introduces less visible perturbations than OP, resulting in adversarial images that are more similar to the original ones, although it spent more time to find adversarial images while targeting bigger images.

Comparison with Square Attack and Pixle Here, we discuss and compare results of IPs with PXand SQ. In specific, we chose the restart-iterative as search algorithm of PX, with 20 restarts and 5 iterations. Furthermore, we select as patch size the 1% of pixels of the target-image, and the random function as mapping one. For what pertains to SQ, we set the maximum number of iterations to 20.

As Table 3.7 depicts, it is clear that the SotA-attack with the highest success-rate is SQ, an it is also the least time-consuming attack. Indeed, on average, SQ queries Resnet-24 and the ResNet-50C 10 and 1 times, respectively, while on 224×224 images requires at least 12 queries to fool the model. However, it is the most-visible attack: indeed, on average, it perturbs on average 787 pixels in order to fool DNNs trained on 32×32 images, meaning that it perturbs approximately 76% of the pixels in the target image, to fool such DNNs. Hence, on average, SQ perturbs 10 times the number of pixels modified by IPs. Finally, the PSNR is on average lower

than 30 dB, meaning that the perturbation is visible to human-eye. For what pertains to PX, is success-rate is almost always higher than IPs, except for the MobileNet-v3 case and, as for SQ, it is less time-consuming than IPs, requiring on average 101 queries to fool each DNN. However, as for SQ, PX perturbs more pixels than IPs – on average 5 times the number of pixels modified by IPs –, and the final perturbation is visible to human-eye, since the average PSNR is always lower than 30 dB.

3.2.3 Discussion

The adversarial attack we proposed, i.e., IPs, improves OP from different perspectives. First, it perturbs pixels within the input image by carefully select the perturbation, in order to modify the input image as little as possible, making the adversarial image indistinguishable from it. Second, it modifies an increasing number of pixels, rather than a fixed one. Such improvements allow IPs having a higher success-rate against different DNNs trained on different dataset, rather than OP. Finally, IPs is able to craft an adversarial sample with a number of queries that is comparable to the one of OP, although it introduces less perturbation.

However, IPs has some drawbacks. IPs selects the pixel to perturb by leveraging the DE genetic algorithm. Such a choice 1. introduces a source of randomness within the attacks. Thus, running IPs twice on the target image could generate different adversarial counterparts. 2. Second, IPs could waste time in modifying pixels that affect less the final classification, meaning that the number of queries could increase. A solution could be selecting the pixel to modify differently, for instance by evaluating in a black-box manner how the modification of a single pixel, or a group of pixels, affects the final classification. However, such solution could increase the number of times the attack queries the DNN.

Furthermore, once selected a pixel and a perturbation, such a perturbation is applied to all the channels of that pixels, and the sign of the perturbation

(for each channel) is decided by evaluating the mean value of the surrounding pixels. However (i) it could be not useful to apply the perturbation to all the channels of a pixel. A solution could be perturbing only some channels, or applying different perturbations for each channel. Furthermore, (ii) mean value is influenced by the outlier, resulting in the potential for a pixel to be either excessively or insufficiently perturbed.

Finally, although the IPs attack is designed to minimize perceptual distortion and evade visual inspection, its sparse and localized perturbations may be more susceptible to detection by automated anomaly detectors. Future work will explore the detectability of IPs under various defense mechanisms.

4

The Effects of Approximate Computing on Adversarial Machine Learning

This chapter investigates the interplay between the Approximate Computing (AxC) design paradigm and adversarial machine learning, with the goal of understanding how efficiency-oriented design choices affect the security of neural networks. While AxC has been proposed as a promising paradigm to reduce the computational and energy footprint of Deep Learning (DL) models, the modifications it introduces to the internal representation of networks may alter their robustness to adversarial perturbations. Despite the growing interest in this area, the relationship between approximation, adversarial vulnerability, and defensive strategies has not yet been fully clarified.

Indeed, as outlined in Section 2.2 of Chapter 2, Convolutional Neural Networks (CNNs) are inherently vulnerable to adversarial attacks, and several defense strategies have been proposed to counteract them. Such approaches generally aim to either (i) increase the robustness of the models, making them less susceptible to adversarial perturbations, or (ii) detect adversarial inputs at inference time in evasion scenarios. At the same time, efficiency-oriented methodologies – such as quantization and AxC – have been shown to influence adversarial robustness. However, the specific impact of AxC on this phenomenon remains largely unexplored.

In this chapter, we address this gap through a systematic study that combines different perspectives.

- First, we analyze the vulnerabilities of Approximate Neural Networks (AxNNs) when exposed to a wide range of adversarial attacks [165]. We show that, although AxNNs remain vulnerable, they tend to be more resilient against gradient-based attacks due to the distortion introduced by approximation. Such an analysis contributes to address the Challenge 2.

- Second, we investigate whether AxC can be leveraged as a source for transferable adversarial examples, demonstrating that adversarial samples generated on AxNNs can be successfully transferred to different target models [166]. This contribution addresses the Challenge 4.

- Finally, we examine the effectiveness of defenses based on input transformations, which are commonly adopted to detect adversarial examples [167], allowing us to address the Challenge 3.Our results indicate that these methods, while effective for floating-point networks, lose much of their discriminative power in approximate and quantized settings.

In order to support these analyses, we also describe the methodology [168] adopted to approximate CNNs, including the design space exploration and the substitution of exact multipliers with approximate ones. This methodological step is essential to ensure that the evaluation is consistent across the three contributions.

4.1 Inspect-NN

As detailed in Chapter 2, the AxC design paradigm improves the efficiency of CNN-based application by typically substituting exact multipliers [150, 133]

within CNNs convolutional and fully connected layers with approximate ones – due to their high hardware requirements [135, 136] –. However, quantifying the error introduced by these multipliers (in the deployment phase) requires extensive hardware prototyping. Hence, software emulators of approximate CNNs are executed on Central Processing Units (CPUs) or General Purpose - Graphic Processing Units (GP-GPUs). However, since the both CPUs or GP-GPUs lack hardware support for approximate arithmetic operations – multiplications –, the CNN emulators are slow. As a consequence, the approximate multiplications also have to be emulated.

To address such a problem, the Inspect-NN framework [168] both provides an efficient emulation for approximate multipliers, and designs AxNNs. Such a framework (i) implements approximate multipliers as look-up tables accessed trough the memory mechanism if CUDA-capable GP-GPUs, and (ii) provide approximate implementations of CNN that minimize both the accuracy loss – due to the approximate multipliers – and the hardware overhead through a Multi-Objective Optimization (MOO) based Design Space Exploration (DSE), that is, a Multi-objective Optimization Problem (MOP).

In specific, Inspect-NN generates AxNNs by replacing 8-bit integer multipliers from the EvoApproxLib library [148] to the original ones in the convolutional and fully-connected layers of the target CNN. For this reason, the target CNN has first to be quantized, in order to represent the floating-point weights into the 8-bit integer representation. Subsequently, Inspect-NN approximates the Quantized Neural Network (QNN) by replacing the exact multiplications performed by neurons with 8-bit approximate ones.

4.1.1 Workflow

Figure 4.1 describes in detail the workflow for approximating the CNNs. In specific, the network is first trained by taking into account the quantization, in order to have a quantized model (with integer parameters) that preserve most of the accuracy of the Full-Precision Neural Network (FpNN). Subsequently,

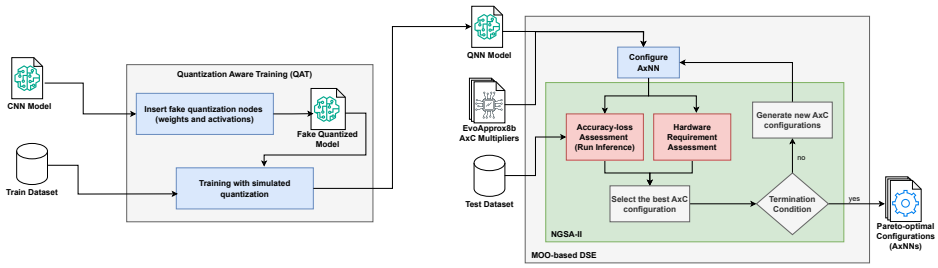


Figure 4.1: Methodology for approximating the CNNs.

Inspect-NN search for a suitable configuration of approximate multipliers that generate an AxNN that (i) preserves as much accuracy as possible of the quantized network, and (ii) optimizes the hardware overhead.

First Step: Quantization

The first step involves the Quantization-Aware Training (QAT) [124] of the target CNN. Such a technique integrates the quantization process in the training process itself, enabling the network to adapt to quantization noise and to learn robust representations that remain accurate under integer-only inference. Such a quantization method relies on an *affine quantization mapping*, that maps real values r, i.e., weights or activations, into integers q, as in Equation (4.1)

$$r = S \cdot (q - Z) \tag{4.1}$$

Here, $S \in \mathbb{R}^+$ is the *scale factor*, defining the quantization step size, while $Z \in \mathbb{Z}$ is the *zero point*, i.e., the integer corresponding to the real value zero. In such a way, the zero is exactly representable, which simplifying operations and reduces cumulative quantization errors.

The QAT modifies the forwarding pass of the training by inserting fake quantization nodes into the computational graph of the CNN. First, weight are quantized before being used in convolutional or fully connected layers,

according to their dynamic range, by mapping their minimum and maximum values to the available integer range (static quantization). Second, activations are quantized after non-nonlinearities, e.g., RELU, or residual connections to simulate inference-time behavior. In this case, ranges are not fixed but estimated during training (dynamic quantization). Subsequently, after quantized weights and activations, the forwarding pass is executed in floating-point arithmetic, which mimics the rounding and clamping that would occur in integer-only inference (fake quantization). Finally, in the backward pass, gradients are computed w.r.t. the underlying floating-point parameters. This avoids the accumulation of quantization error and allows for precise weights updates. Formally, the quantization function applied during the forward step can be expressed as in Equation (4.2).

$$q(r;a,b,n) = \left\lfloor \frac{\text{clamp}(r;a,b) - a}{s(a,b,n)} \right\rceil \cdot s(a,b,n) + a \tag{4.2}$$

$$s(a,b,n) = \frac{b-a}{n-1} \tag{4.3}$$

$$\text{clamp}(r;a,b) = \min\big(\max(r,a), b\big) \tag{4.4}$$

Here, r s the real value (weight or activation) to quantize, while $[a,b]$ is the quantization range, e.g., $[0,255]$ for integer 8-bit positive-only quantization. The $\text{clamp}(r;a,b)$ operation is a clipping operation that forces the real value r to be mapped in the range $[a,b]$. Indeed, $\text{clamp}(r;a,b) = \min(\max(r,a),b)$ n is the number of quantization levels, e.g., 256 for 8-bit quantization. Finally, $s(a,b,n) = \frac{b-a}{n-1}$ is the quantization step, while $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.

Despite QAT is computationally more intensive than Post-Training Quantization (PTQ), (i) it allows for a better preservation of the accuracy, and (ii) QAT models can be can be directly exported as quantized networks, with

integer-only weights and activations, ready for execution on constrained hardware [122, 123, 169].

Second Step: Approximation

In the second step, the QNN is quantized by solving a MOO-based DSE problem. Inspect-NN first identifies the approximable layers, i.e., convolutional and fully-connected ones. The goal is to find one – or more – configuration of approximate multipliers, one per layer, that both minimizes the silicon area and the power consumption (once deployed the AxNN on the hardware) by constraining the accuracy-loss. To do so, the MOO-based DSE is encoded by using a set of decision variables for each layer to be approximate: the value of each of them specifies the approximate operator to be used in the corresponding layer. The range of decision variables clearly depends on the number of possible alternative multipliers that can be used and, together with the number of layers, the size of the solution space. The solution of such a problem is called *feasible-solution*, that is, a solution that optimize the *fitness-metrics* (hardware overhead, such as the silicon area and the power consumption) and met the constraints. Since this MOP is not-trivial, there are multiple feasible solutions. Among this, we search for the *Pareto-optimal solutions*, i.e., feasible solutions that cannot be further improved in any of the objectives (fitness-metrics) without degrading at least one of the other objectives, or without violating at least one constraint. However, since the dimension of the design space rapidly grows with the number of decision variables, fitness functions, and constraints, we solve such a hard problem with the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [170] meta heuristic. In this context, approximate implementations of a given function are encoded as a n-genes chromosome, with one gene for each of the layers to be approximated individually.

During the DSE, the error introduced due to the approximation is computed before a new candidate solution has to be evaluated.This is the most

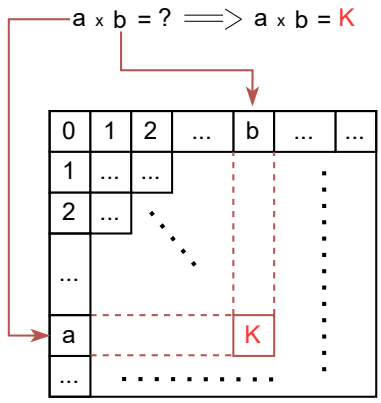


Figure 4.2: Lookup-based multiplication

demanding step in terms of computation time. Indeed, performing the inference phase of an AxNN requires emulating such components which slow down the computations by several orders of magnitude w.r.t. the reference software implementation operating with standard floating-point/integer arithmetic. To address this, we replace each layer to be approximate with our custom implementation, that allows for the specification of the approximate multiplier to be emulated for that specific layer. During the inference phase, the output volume of each layer is computed following the sequence defined by the network architecture. However, when the output volumes need to be computed while relying on approximation operators, our custom implementation is executed instead of the one provided by common machine learning frameworks.

The functional behavior of approximate components with look-up tables, as in Figure 4.2. Each different component is modeled using a table that maps the output corresponding to any input combination. Operands of the multiplication behave as row and column indexes to access the table and to select the corresponding output value. Look-up tables are stored into Graphic Processing Unit (GPU) texture cache, thereby avoiding access conflicts. Pertaining to the error assessment procedure, Inspect-NN exploits the parallelism provided

by the GP-GPUs: the inference phase is split into blocks, each executed by an independent thread block on the GP-GPU. Inspect-NN performs these computations through a kernel, i.e., a CUDA function called by the CPU and executed on the GP-GPUs. Operations within each layer are actually parallelized such that each thread block executes a fraction of the overall operation. Thread-reuse is employed to minimize the scheduling overhead, and dynamic grid assignment is exploited so that threads are all guaranteed to be executing concurrently. Furthermore, data exchange between the CPU and the GP-GPU is reduced to the bare minimum: data are copied from the GP-GPU memory to the CPU once, when strictly required. Hence, if two consecutive layers are working on the GP-GPU, the first one feeds the GP-GPU memory address of the computed data to the next layer, rather than copying them back and forth from/to the CPU.

Finally, since Inspect-NN implements a look-up table version of the 8-bit integer multipliers from the EvoApproxLib library – note that that the look-up table version is used to generate the AxNNs – Table 4.1 reports the error characterization and hardware requirements of the multipliers. For further details and a comprehensive explanation of the multipliers and their error characterization, as well as their hardware overhead, the reader is referred to [148].

4.2 Experimental Testbed

Before presenting the three contributions of this chapter, it is important to introduce the experimental testbed adopted throughout the analyses. Since all contributions share the same datasets, network architectures, and implementation variants, as well as common training strategies, this section provides a unified description of the environment in which all experiments were conducted. This avoids redundancy across the following sections and ensures that the results presented later can be interpreted under a consistent framework.

Circuit	**MAE** (%)	**AWCE** (%)	**MRE** (%)	**Power (mW)**	**Area** (μm^2)
1KV6	0.00	0.00	0.00	0.425	729.8
1KV8	0.0018	0.0076	0.28	0.422	711.0
1KV9	0.0064	0.026	0.90	0.410	685.2
1KVA	0.019	0.075	2.53	0.391	641.1
1KVM	0.049	0.20	2.40	0.369	652.8
1KVP	0.051	0.21	2.73	0.363	635.0
1KVQ	0.056	0.25	3.64	0.351	599.8
1KX5	0.15	0.69	8.93	0.289	543.0
1KXF	0.34	1.37	15.72	0.237	482.4
1L2J	0.081	0.39	4.41	0.301	558.9
1L2L	0.23	1.16	12.26	0.200	411.6
1L2N	0.52	2.66	27.44	0.126	284.9
1L12	3.08	12.30	135.77	0.052	172.2

Table 4.1: Error characterization and hardware requirements for approximate circuits taken from the EvoApproxLib-Lite library, as reported in [148].

In particular, we first describe the dataset used for both training and evaluation, and we introduce the CNNs architectures that serve as baselines for our study. We then outline the different implementations considered—floating-point, quantized, and approximate—and summarize their performance in terms of classification accuracy and estimated hardware requirements. Finally, we detail the training procedure and the computational resources employed during the experiments.

This unified setup forms the foundation for the subsequent sections: Section 4.3 analyzes the robustness of AxNNs against adversarial attacks, Section 4.4 explores the transferability of adversarial examples crafted on approximate models, and Section 4.5 investigates the effectiveness of input transformation defenses.

To assess the impact of AxC on adversarial robustness, we selected three CNNs of increasing complexity. The first is MinNet, a custom lightweight model specifically designed for resource-constrained environments, whose architecture is reported in Table 4.2. It consists of 7 layers, to which no stride or padding are applied, and it exploits the ReLU activation function. Having only 350 thousand trainable parameters, MinNet is fairly small when

compared to State-of-the-Art (SotA) CNNs targeting the CIFAR-10. The second and third are ResNet8 and ResNet24, two residual networks of different depths from the ResNet family [155] that allow us to analyze the behavior of approximation in more complex architectures. Together, these models cover a spectrum from shallow, highly efficient CNNs to deeper and more accurate ones, making them a representative testbed for our study.

Table 4.2: Architectural details of the MinNet CNN.

Layer	Input Shape	Output Shape	# of Parameters
Conv2D	1x32x32x32	1x30x30x32	896
MaxPooling2D	1x30x30x32	1x15x15x32	0
Conv2D	1x15x15x32	1x13x13x64	18496
MaxPooling2D	1x13x13x64	1x6x6x64	0
Flatten	1x6x6x64	1x2304	0
Dense	1x2304	1x64	147520
Dense	1x64	1x10	650

For each network, we considered three distinct implementations. The floating-point version (FpNN) represents the baseline, trained with 32-bit precision, designed and trained with the TensorFlow Python library [171]. All floating-point versions were trained for 200 epochs using the Adam optimizer and the cross-entropy loss. We used a batch size of 64 and standard data preprocessing (normalization and random data augmentation). A quantized version (QNN) was obtained through Quantization-Aware Training (QAT), reducing both weights and activations to 8-bit integer precision while attempting to preserve accuracy. The QNNs were implemented with TensorFlow-Lite [172]. Finally, approximate versions (AxNNs) were generated with Inspect-NN [168], which systematically replaces exact multipliers in convolutional and dense layers with approximate ones drawn from the EvoApprox8b library [148]. Approximation was guided by a multi-objective optimization procedure aimed at striking a balance between accuracy degra-

dation and hardware efficiency, measured in terms of power consumption and silicon area.

Table 4.3: Accuracy, and estimated resource requirements for MinNet, ResNet8 and ResNet24 implementations.

Model	**Accuracy (%)**	**Power (mW)**	**Area (μm^2)**
MinNet	80.08	125.27	4860.35
Quantized MinNet	78.80	26.20	2919.2
Approximate MinNet v1	75.02	14.73	1990.2
Approximate MinNet v2 [166]	78.84	15.23	1990.2
ResNet8	86.36	1390.12	13365.97
Quantized ResNet8	84.32	288.48	8027.8
Approximate ResNet8 v1	81.61	175.01	6043.6
Approximate ResNet8 v2 [166]	85.89	202.30	6451.99
ResNet24	84.39	2487.87	25516.85
Quantized ResNet24	86.50	723.80	15325.80
Approximate ResNet24 v1	87.59	544.96	12575.09
Approximate ResNet24 v2 [166]	88.17	537.80	12276.89

Table 4.3 reports details regarding each considered CNN. For such networks, both the accuracy and resource-requirements, i.e., the estimated area occupied on the silicon and the estimated power required to perform an inference, differ from the original floating-point FpNNs to both its quantized and approximated versions. Note that we estimate the hardware overhead of each implementation as done in [173]. Although accuracy is slightly compromised in the case of MinNet and ResNet8, both area and power consumption benefit greatly from the quantization and approximation of the networks. In addition, note that for each CNN, we considered two different approximate versions (vi and v2) only for the experiment of [166], whose results are discussed in the Section 4.4. For all the other cases, we considered only one approximate version, i.e., the v1 one. Finally, note that, from now on, we will refer for semplicity to: (i) the floating point MinNet, ResNet8, and ResNet24 as FpM, FpR8, and FpR24, respectively; (ii) the quantized MinNet, ResNet8, and ResNet24 as QM, QR8, and QR24, respectively, and (iii) the approximate ver-

sions of MinNet, ResNet8, and ResNet24 as AxC0M and AxC1M, AxC0R8 and AxC1R8, and AxC0R24 and AxC1R24, respectively

All experiments were conducted on the CIFAR-10 dataset [156], which contains 60 thousand 32×32 RGB images equally distributed across 10 classes. The dataset was partitioned in order to allocate 60% of the images for training purposes, 20% to validate the training process, and reserve the remaining 20% for use as test dataset. This choice ensures that adversarial evaluation is not biased by misclassified clean samples, and provides a fair ground for comparison across different implementations. In addition, note that, since the experiment were conducted in an evasion scenario, among the test images $x \in X$, we select only the ones that were correctly classified by all the implementations of a network. This means that, given a specific test image x and its true label y, and given three inference functions f_{fp}, f_q, and f_{axc} for the FpNN, QNN, and AxNN, respectively – in this case, we have 3 different implementations for a given CNN – , such an image was used in the experiment only if the condition in Equation 4.5 is met.

$$\arg\max(f_{fp}(x)) = \arg\max(f_q(x)) = \arg\max(f_{axc}(x)) = y \tag{4.5}$$

For what pertains adversarial-attacks generation, to the best of our knowledge none of the SotA frameworks, such as CleverHans [174] or Adversarial Robustness ToolBox (ART) [175], do allow neither targeting quantized and approximate neural networks, nor observing the effort required to craft successful adversarial-samples. Therefore, we re-implemented algorithms for adversarial-attacks generation either starting from zero or by leveraging the implementations provided by ART.

Finally, the experiments were carried out on a workstation equipped with an AMD Ryzen Threadripper PRO 5945WX CPU and an NVIDIA RTX A5000 GPU. All implementations and adversarial evaluations were developed in Python using TensorFlow 2.12.

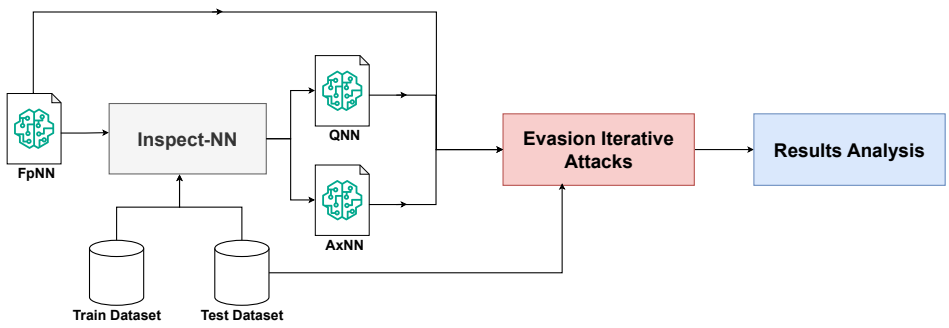


Figure 4.3: The methodology we developed to evaluate the robustness of AxNNs against adversarial attacks.

4.3 Improving the Robustness of CNNs with Approximate Computing

The first contribution addresses the vulnerabilities of AxNNs w.r.t. adversarial attacks [165]. While the AxC design paradigm is often considered to exploit the error resilience of deep models to enhance their hardware efficiency, its effect on adversarial robustness remains ambiguous. Indeed, Guesmi *et al.* [144] propose the AxC as a defense method against adversarial attacks, altough their approximation technique still leverage floating-point arithmetic circuits. Conversely, Siddique and Hoque [147] found that AxNN are more vulnerable to adversarial attacks than their original floating-point counterpart. Also, they found that the AxC decreases the adversarial robustness gained trough the quantization. Finally, Atoofian [149] increased the robustness against adversarial attacks by building an ensemble of AxNNs, each of them approximate with a single approximate circuits.

Therefore, the Challenge 2 is still unsolved, due to such conflicting results from the SotA. Trough this contribution, the key question investigated is whether AxC can provide any inherent robustness to adversarial attacks, or whether AxNNs remain as vulnerable as their floating-point counterparts.

To answer this question, we trained several CNNs – both custom and from the SotA – and generated their quantized and approximate versions. The latter AxNNs were generated by leveraging Inspect-NN, as described in Section 4.1. Subsequently, we evaluated the robustness of the FpNNs, QNNs, and AxNNs under a set of adversarial attacks from the SotA. In order to obtain the most comprehensive analysis possible we evaluated a series of attacks belonging to different typologies.

- Gradient-based iterative white-box attacks, which iteratively exploit the gradient of the loss function to craft perturbations. Of course, such attacks require a complete knowledge about the target CNN.

- Optimization-based white-box attacks. As the gradient based attacks, these require a complete knowledge about the target CNN. However, optimization-based attacks aim to find minimal perturbations, to apply to the input image, to cross the decision boundary.

- Black-box query-based attacks: they require access to the inputs and outputs of the target model. The adversarial image is crafted by iteratively perturbing the input, and querying the model with such a modified image to verify wether a misclassification has occurred.

Each attack was executed in an evasion scenario, targeting only the test-set samples that were correctly classified by all the implementation of the target CNN – the FpNN, the QNN, and the AxNN –. For each attack, we evaluate the success rate – defined as the percentage of adversarial samples that caused a misclassification –, the perturbation, and the number of queries required to craft a successful adversarial image. Figure 4.3 depicts the whole methodology.

The rationale behind this analysis is that approximation and quantization alter the smoothness of the loss landscape, thus limiting the effectiveness of gradient-based attacks that rely on precise differentiability. On the other hand,

black-box and optimization-based methods do not rely on gradients, and may therefore still succeed against AxNNs.

Indeed, white-box attacks target exclusively the FpNN, since they are gradient-base and they require the differentiability of the loss function. Subsequently, each successful adversarial image is transferred from the floating-point implementation to both the quantized and approximate version of the target CNN. Conversely, white-box attacks do not require the differentiability of the loss function, thus allowing us to target every implementation of the target CNN.

4.3.1 Evaluation Metrics

In order to evaluate the result once targeted the CNNs with adversarial attacks, we employed the following evaluation metrics:

- *Success rate*, that is, the proportion of adversarial examples that successfully cause the target model to misclassify with respect to the total number of attempted adversarial perturbations.
- *Transferability-rate*, that is, the proportion of adversarial examples that are transferable from an implementations to the others of a CNN.
- The *perturbation-strength*, that is, the total perturbation δ_x applied to the original image x in order to compute an adversarial image.
- *Number of queries*: we evaluate how many times each attack-algorithm queries the target implementation, in order to estimate the computational complexity while generating an adversarial image.
- *Peak Signal-to-Noise Ratio (PSNR)* [176]: it is a quality-metric that quantifies the degree of distortion of a noisy (perturbed) image w.r.t. the original image. It is computed by considering the ration between the maximum value that a pixel can assume, e.g, 255 in case of 8-bit

RGB pixels, and the Mean Square Error (MSE) between the original image x and the perturbed one x', as depicted in Equation (4.6).

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}(x, x')} \right) \tag{4.6}$$

4.3.2 Experimental Evaluation

In order to evaluate the robustness of AxNNs w.r.t. adversarial attacks, we targeted the CNNs whose details are depicted in Table 4.3 of section 4.2. Note that, for such a contribution, we considered only the v1 approximate implementation of MinNet, ResNet8, and ResNet24. For a complete analysis, we targeted the CNNs with both white-box and black-box attacks from the SotA. For what pertains to the adversarial attacks, we targeted the CNNs with Basic Iterative Method (BIM), Projected Gradient Descent (PGD), Carlini-Wagner (CW), DeepFool, and One-Pixel (OP). In specific, each white-box attack – note they are all iterative attacks – applies an incremental perturbation using a step size of $1/255$ for each iteration. The reason is that the test images of CIFAR-10 are 32×32 ones, normalized to 8-bit positive integers. This means that each channel of each pixel as a value in the range [0,255]. For what pertains to the number of iterations, we set to 50 the maximum number in case of BIM, PGD, and DeepFool, and to 20 in case of CW. Concerning the black-box OP, the perturbation is applied randomly, as in the original paper [25], although our implementation is an incremental one. Finally, we set to 97 and 60 the number of generations and the population size of Differential Evolution (DE) – the genetic algorithm leveraged by OP – In case of MinNet. Conversely, we set to 80 the populations size in case of both ResNet8 and ResNet24, while the number of generations is set to 97 in case of ResNet8 (as for MinNet), and to 51 in case of ResNet24.

Table 4.4: Success rate of BIM, PGD, CW, DeepFool, and OP against the implementations of MinNet, ResNet8, and ResNet24. Note that, since white-box attacks require the differentiability of loss-function, they are not applicable to the quantized and approximate implementations.

CNN Implementation	Success Rate				
	BIM	PGD	CW	DeepFool	OP
FpM	100	100	34.80	75.84	75.73
QM	-	-	-	-	77.17
AxCMv1	-	-	-	-	75.92
FpR8	99.83	99.83	17.08	71.26	66.16
QR8	-	-	-	-	51.26
AxCR8v1	-	-	-	-	27.29
FpR24	99.42	99.42	14.37	69.63	58.04
QR24	-	-	-	-	43.52
AxCR24v1	-	-	-	-	29.56

Table 4.5: Transferability rate from the floating-point implementation to the quantized and approximate ones, for MinNet, ResNet8, and ResNet24 of BIM, PGD, CW, DeepFool, and OP.

Source Implementation	Target Implementation	Transferability Rate				
		BIM	PGD	CW	DeepFool	OP
FpM	QM	36.16	35.83	71.80	89.34	62.27
	AxCMv1	31.44	33.50	69.48	86.75	59.08
FpR8	QR8	15.77	16.25	51.26	88.83	33.86
	AxCR8v1	15.91	13.93	42.70	88.59	29.56
FpR24	QR24	12.17	14.25	52.37	89.34	29.94
	AxCR24v1	13.16	12.28	49.15	86.75	16.90

Results

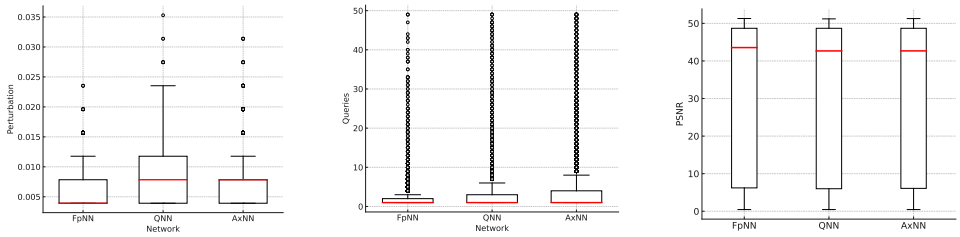
Tables 4.4 and 4.7 show the success and transferability rates of adversarial attacks against MinNet, ResNet8, and ResNet24. Note that, since the white-box attack we considered are gradient-based, they require the transferability of the loss function. For this reason, they are not applicable to the quantized and approximate implementations.

Both BIM and PGD have the greatest success rate, that is 100% against FpM, and greater than 99% against FpR8 and FpR24 – please refer to

Section 4.2 for the notation –. However, the transferability-rate is poor. Indeed, it is in the range $\sim [31,37]\%$ from the floating-point against the quantized and approximate implementations of MinNet, and in $\sim [12,17]\%$ while targeting ResNet8 and ResNet24. CW has the poorest success-rate – $\sim 30\%, \sim 17\%$ and $\sim 14\%$ against FpM, FpR8, and FpR24, respectively–. However, its transferability-rate is in the range $\sim [69,72]\%$ in case of MinNet, and $\sim [42,53]\%$ while targeting ResNet8 and ResNet24. Finally, the success rate of DeepFool is in the range $\sim [69,76]\%$, while its transferability rate is the greatest, that is almost 90%. For what pertains to the black-box scenario, OP has a success rate greater than 75% against all the implementation of MinNet. Nonetheless, the deepest the architecture, the lowest is the success-rate. Indeed, it is in the range $\sim [58,66]\%$ while targeting the floating-point implementations of ResNet8 and ResNet24. Conversely, the success-rate against the quantized and approximate implementations decreases significantly, especially while targeting the approximate implementations. Similarly, the transferability-rate from the floating-point against the quantized and approximate implementations decreases with deeper networks (ResNet8 and ResNet24), and it is even smaller while transferring to AxNNs.

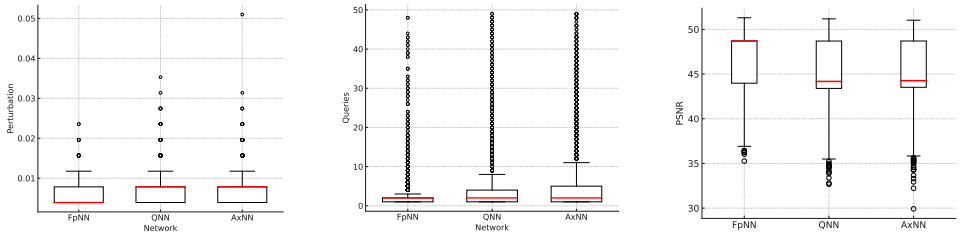
Such results suggest that, while both the success and the transferability rates depends on the architecture of a floating-point CNN, it is not true in case of the quantized and approximate versions. Indeed, the success rate decreases with the precision of the target implementation, since it is lower while targeting the quantized one – it has less-precise parameters – , and even lower while targeting the approximate implementation – it has less-precise parameters and multiplications –. Conversely, the transferability-rate is similar – with negligible differences – while transferring from floating-point to both quantized and approximate implementations, except for the OP case while targeting ResNet24.

Figures 4.4, 4.5, 4.6, 4.7, and 4.8 depict the effort required by BIM, PGD, CW, DeepFool, and OP, respectively, to fool all the implementation of Min-



(a) Perturbation **(b)** Number of queries **(c)** PSNR

Figure 4.4: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by BIM to fool the floating-point, quantized and approximate implementations of MinNet.

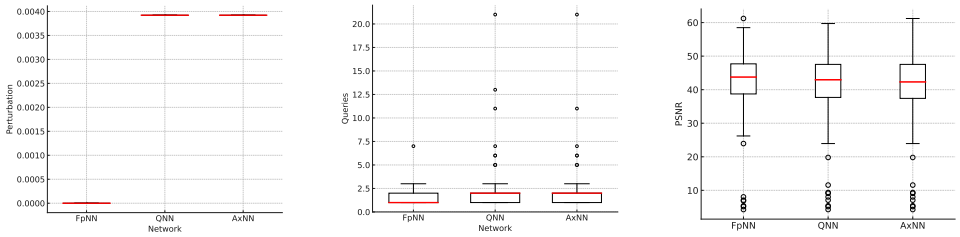


(a) Perturbation **(b)** Number of queries **(c)** PSNR

Figure 4.5: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by PGD to fool the floating-point, quantized and approximate implementations of MinNet.

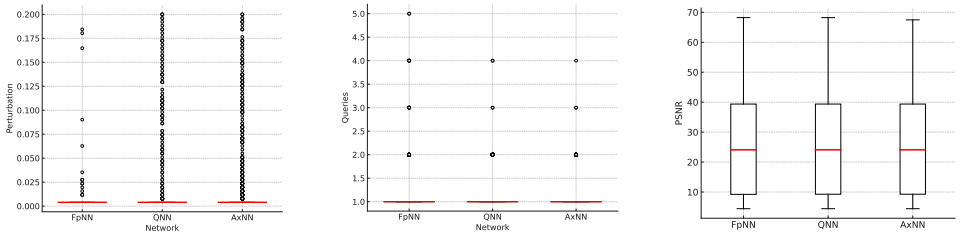
Net. Each attack requires a comparable effort effort (perturbation straighten and number of queries) fool all the implementations of MinNet, although it is slightly higher while targeting the quantized and approximate implementations (in case of BIM, PGD, and CW), and while targeting the AxCM with OP. Indeed, such a conclusion is confirmed by the PSNR boxplots.

Figures 4.9, 4.10, 4.11, 4.12, and 4.13 depict the effort required by BIM, PGD, CW, DeepFool, and OP, respectively, to fool all the implementation of ResNet8. Conversely, Figures 4.14, 4.15, 4.16, 4.17, and 4.18 depict the effort required by BIM, PGD, CW, DeepFool, and OP, respectively, to fool all the implementation of ResNet24.



(a) Perturbation **(b)** Number of queries **(c)** PSNR

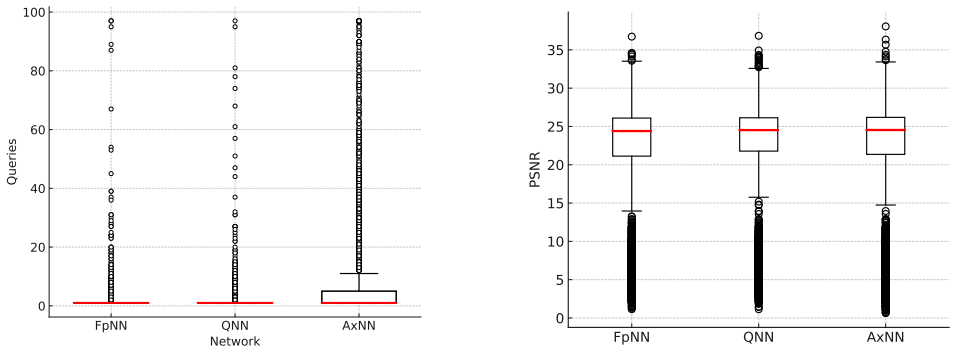
Figure 4.6: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by CW to fool the floating-point, quantized and approximate implementations of MinNet.



(a) Perturbation **(b)** Number of queries **(c)** PSNR

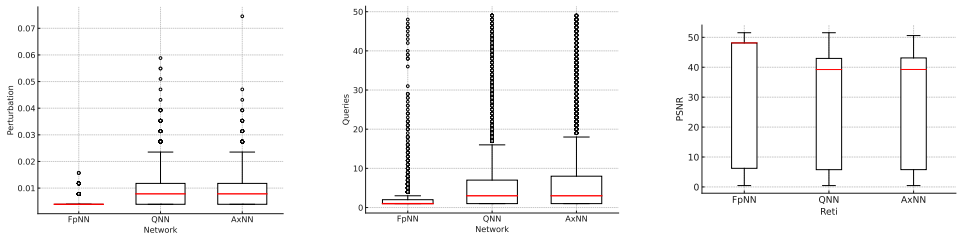
Figure 4.7: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by DeepFool to fool the floating-point, quantized and approximate implementations of MinNet.

All the attacks (except for DeepFool) require always a greater effort to fool the quantized and approximate versions of both ResNet8 and ResNet24, in terms of both the alteration – perturbation – added to the original mages and the number of queries required to fool such implementations. This is also confirmed by the PSNR boxplots. Indeed, the PSNR is always lower while targeting the quantized and approximate implementations than that of the floating-point one. Finally, as for MinNet, the effort is greater exclusively while targeting the approximate implementations. Such results suggest the approximate implementations are never less robust against adversarial attacks than floating point and quantized implementation. Indeed, the effort required



(a) Number of queries **(b)** PSNR

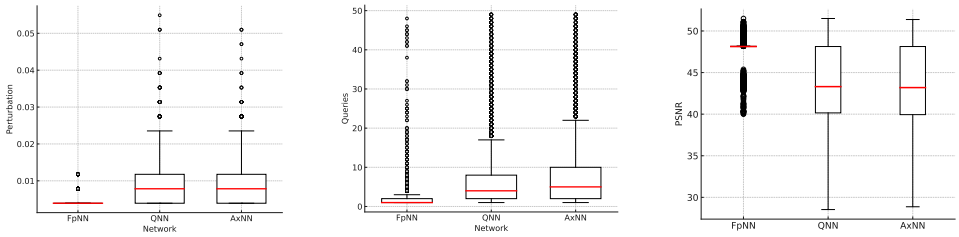
Figure 4.8: Number of queries **(a)**, and PSNR **(b)**, required by OP to fool the floating-point, quantized and approximate implementations of MinNet.



(a) Perturbation **(b)** Number of queries **(c)** PSNR

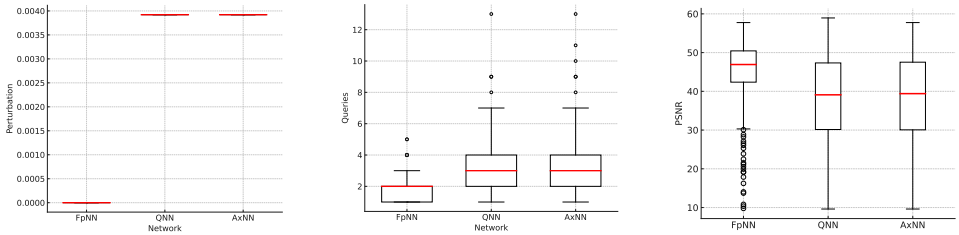
Figure 4.9: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by BIM to fool the floating-point, quantized and approximate implementations of ResNet8.

to fool the AxNNs is always similar to and greater than the ones required to fool the QNNs and FpNNs, respectively, for shallow networks. Conversely, for deeper networks, the effort required to fool the AxNNs is always greater than the ones required to fool both the FpNNs and QNNs, in contrast to what claimed by Siddique *et al.* [147].



(a) Perturbation **(b)** Number of queries **(c)** PSNR

Figure 4.10: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by PGD to fool the floating-point, quantized and approximate implementations of ResNet8.

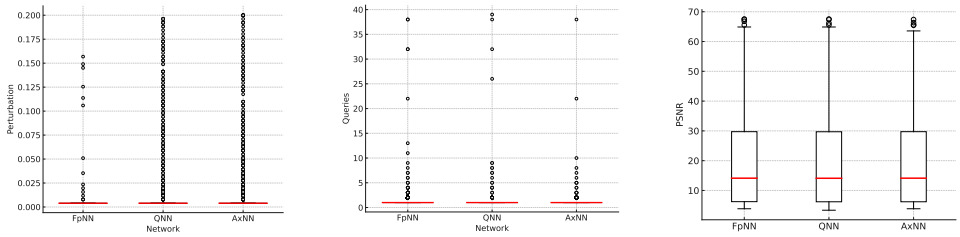


(a) Perturbation **(b)** Number of queries **(c)** PSNR

Figure 4.11: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by CW to fool the floating-point, quantized and approximate implementations of ResNet8.

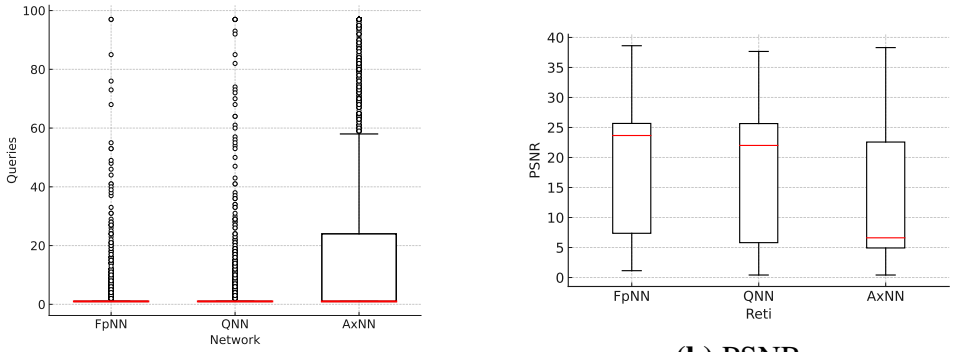
4.3.3 Discussion

With this contribution we solved the Challenge 2. Indeed, this contribution answers in a complete and definitive manner wether the AxC is a defense strategy against adversarial attacks. In the specific case of CNNs approximate with 8-bit integer multipliers, and a specific approximate multiplier per layer, the AxC improves the robustness of CNNs w.r.t. adversarial attacks. Such AxNNs are not vulnerable to gradient-based adversarial attacks, since their loss-function is not differentiable. Furthermore, the AxC drastically reduces the the success-rate of query-based adversarial attacks, especially in case of deep architectures, as well as the transferability-rate from FpNNs to AxNNs.



(a) Perturbation **(b)** Number of queries **(c)** PSNR

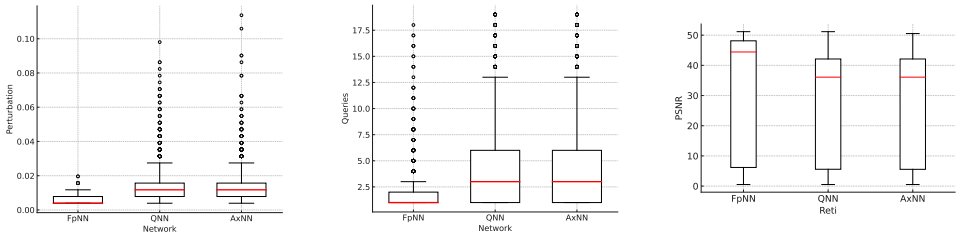
Figure 4.12: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by DeepFool to fool the floating-point, quantized and approximate implementations of ResNet8.



(a) Number of queries **(b)** PSNR

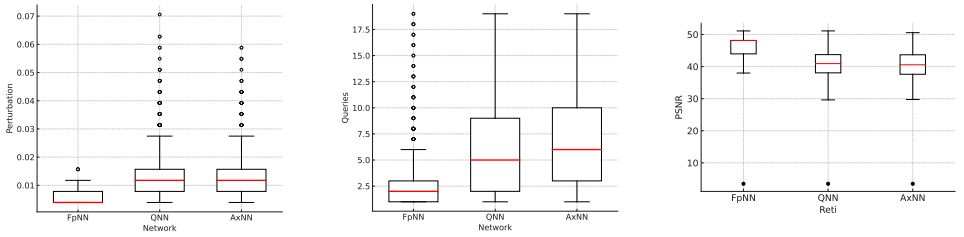
Figure 4.13: Number of queries **(a)**, and PSNR **(b)**, required by OP to fool the floating-point, quantized and approximate implementations of ResNet8.

Finally, in order to craft successful adversarial against AxNNs, adversarial attacks require a greater effort to fool such a network, in terms of the perturbation applied to the original images, and the number of queries to the approximate network itself.



(a) Perturbation **(b)** Number of queries **(c)** PSNR

Figure 4.14: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by BIM to fool the floating-point, quantized and approximate implementations of ResNet24.

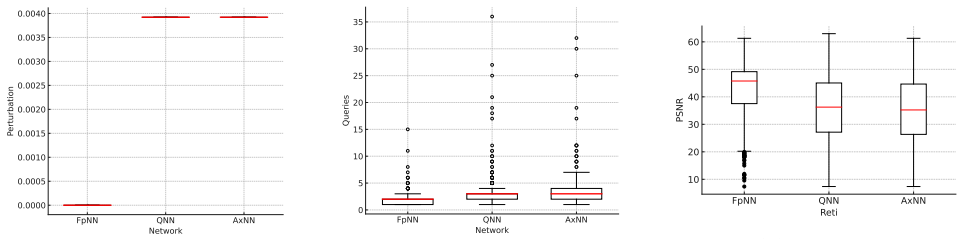


(a) Perturbation **(b)** Number of queries **(c)** PSNR

Figure 4.15: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by PGD to fool the floating-point, quantized and approximate implementations of ResNet24.

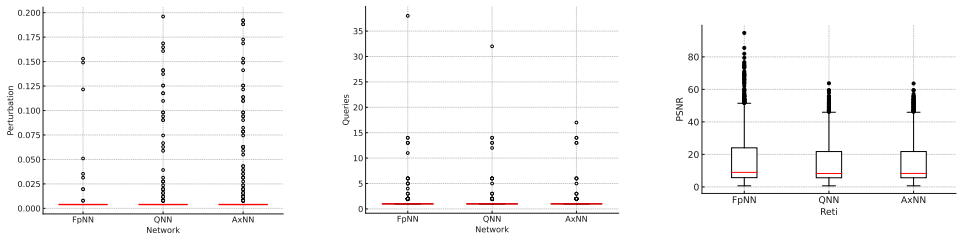
4.4 Transfer Adversarial Attacks trough Approximate Computing

The second contribution investigates the role of AxC in the transferability of adversarial attacks [71]. While transferability is a well-known phenomenon in adversarial machine learning, adversarial samples crafted on one model often remain effective against other models. However, the specific case of AxNNs has been largely neglected in the literature. Our objective is to understand whether approximation, besides reducing the computational cost of CNNs,



(a) Perturbation **(b)** Number of queries **(c)** PSNR

Figure 4.16: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by CW to fool the floating-point, quantized and approximate implementations of ResNet24.

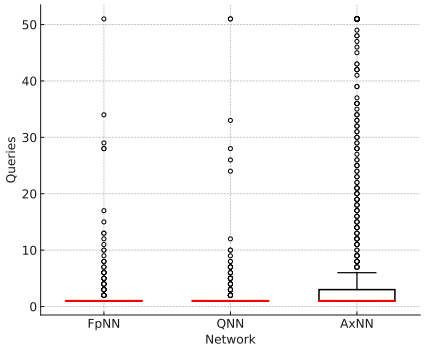


(a) Perturbation **(b)** Number of queries **(c)** PSNR

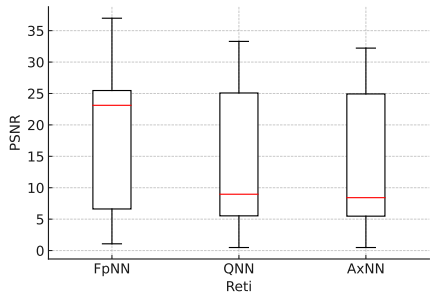
Figure 4.17: Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by DeepFool to fool the floating-point, quantized and approximate implementations of ResNet24.

can also influence the capability of adversarial examples to generalize across different targets [166], in response to Challenge 4.

Specifically, we first generate AxNNs with Inspect-NN, as detailed in Section 4.1. Subsequently, we target an AxNN with black-box query adversarial attacks, and the successful adversarial sample are transferred to the other implementations of any other CNN. Thois means that the adversarial image are transferred to both the floating-point and quantized implementation of the source CNN (the one being approximate), and to any other implementation – FpNN, QNN, and AxNN – of other CNNs. Finally, we examine the transferability-rate. Figures 4.20 and 4.21 depict an overview of



(a) Number of queries



(b) PSNR

Figure 4.18: Number of queries **(a)**, and PSNR **(b)**, required by OP to fool the floating-point, quantized and approximate implementations of ResNet24.

our proposed methodology. Specifically: 1. we first generate the QNN and the AxNN of a target CNN with Inspect-NN. 2. We then target the AxNN with an incremental perturbation crafted through a black-box query-based adversarial-attack approaches. In other words, an incremental perturbation is added to the original image that is input to the AxNN, until the latter is fooled (Figure 4.20). 3. we transfer successful adversarial samples to other CNNs, regardless their architecture and implementations, and we study the transferability rate. This entire process is repeated for all the images of the clean input dataset (Figure 4.21).

The first step of our methodology involves the approximation of neural networks through Inspect-NN, as detailed in Section 4.1.

Once approximated, we iteratively target the AxNN with a query-based black-box adversarial attacks, as Figures 4.20 depicts. White-box gradient-based attacks are not applicable, since quantization typically makes the loss-function of a CNN not differentiable [177]. Such a black-box attack has to craft an adversarial sample by relying on exclusively the input image, and on the output of the AxNN, that is, the probability vector associated to the input. Moreover, as the adversarial sample generation is an iterative process,

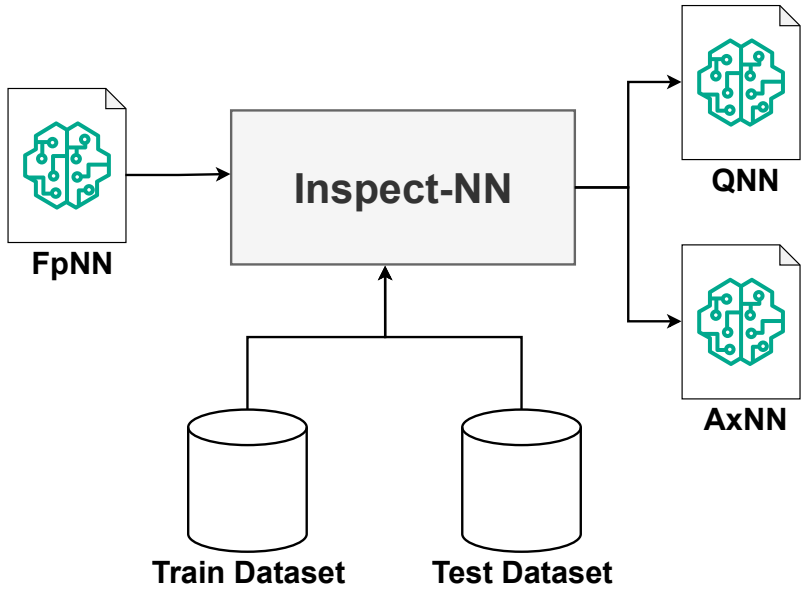


Figure 4.19: Overview of our methodology – Generating the quantized and approximate implementations.

the value of the perturbation—that is, the amount to be added to the input image—must be initialized and then adjusted. The algorithm begins with a minimal perturbation, for instance, either only one pixel must be modified, or the value of the perturbation is $\delta = 1/255$, which is the minimum RGB value applicable to an integer-normalized image. In the event that the adversarial sample is successfully fooled, it is considered a successful adversarial sample. Otherwise, the value of the perturbation must be incremented. This entails modifying more than one pixel or assigning a value greater than $1/255$ to the perturbation. This iterative procedure is iterated until a maximum perturbation is tolerated.

The last step is represented by Figure 4.21. Once a successful adversarial sample of an image (against the AxNN) is generated, we transfer it to other networks, regardless of both the architecture and the implementation of the

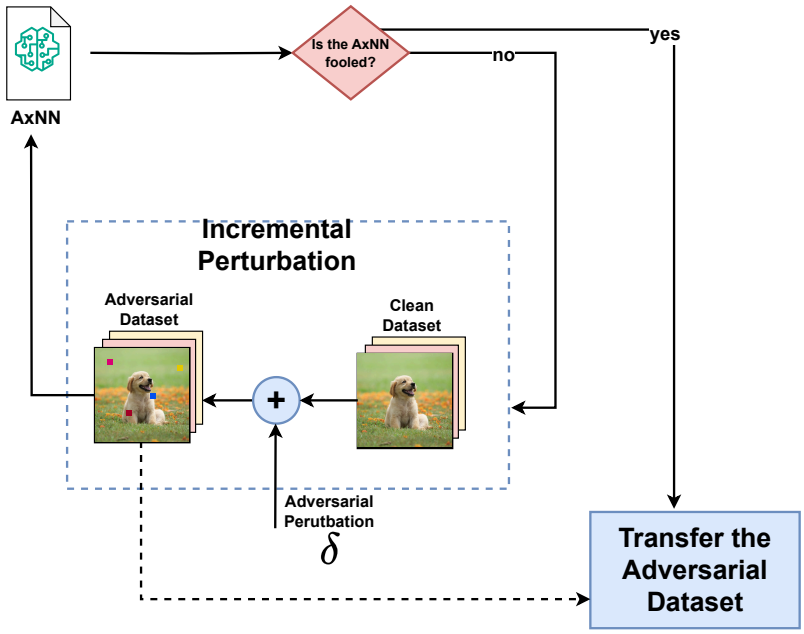


Figure 4.20: Overview of our methodology – Crafting the adversarial dataset.

latter. Specifically, an image is successfully-transferable if, assuming that it fooled the source AxNN, it also fools the new target CNN. Thus, if an image x^{adv} crafted to fool the source AxNN also fool the target CNN, we state that the AxNN successfully generated an adversarial sample against the target network. Finally, we explore the transferability-rate.

Such a process is applied to the whole datasets that contain the images that are correctly classified by the source AxNN and by the implementations of the other networks, to which image are transferred. Furthermore, the adversarial perturbation is computed by querying exclusively the AxNN.

This evaluation framework allows us to study both *intra-architecture* transferability, where the source and target models share the same architecture but differ in implementation (floating-point, quantized, approximate), and

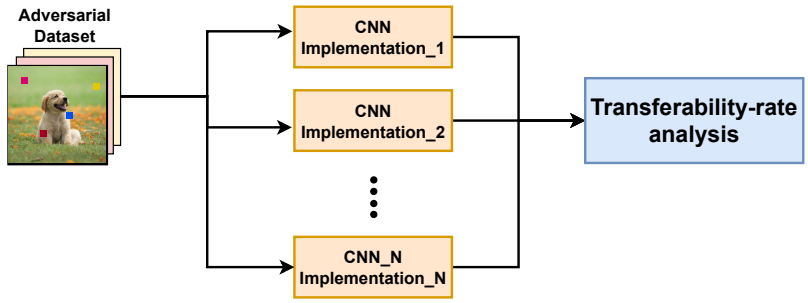


Figure 4.21: Overview of our methodology – Transferring the adversarial dataset.

cross-architecture transferability, where adversarial examples generated on a given CNN are tested on deeper or shallower models.

The hypothesis underpinning this methodology is that the AxC design paradigm, by modifying the decision boundaries of the model, may generate adversarial examples that are more general and thus more likely to transfer to other networks. We also expect the depth of the source and target models to play a crucial role: deeper models may produce adversarial examples with higher transferability, while attacks generated on shallow models may fail to transfer to more complex architectures.

This systematic analysis allows us to quantify to what extent AxNNs can act as effective surrogates for generating transferable adversarial examples, thereby highlighting a potential security risk of Approximate Computing in adversarial settings.

4.4.1 Experimental Evaluation

This Section presents the results of an experimental evaluation conducted with the CNNs and all their respective implementations depicted in Table 4.3 of Section 4.2. Subsequently, an iterative version of the OP attack [25] was used to generate adversarial datasets targeting the approximate implementations

of the neural networks. inally, successful adversarial samples from each adversarial dataset were transferred to the other networks, in order to evaluate the transferability-rate.

Adversarial Datasets Generation

First, note that for our experiments, we considered a subset of 3000 images from the original test dataset X of CIFAR-10, which originally contains 10000 images. Among these images, were considered only the images that undergo the condition 4.5, for each approximate version. This means that, for each CNN, we have two different test subset: the one containing the images correctly classified by the floating-point, quantized, and approximate (v1) implementations, and the other one containing the images correctly classified by the floating-point, quantized, and approximate (v2) implementations. Therefore, we have 1. the X_{AxC0M} and X_{AxC1M} datasets contain 2254 and 2361 images, respectively, 2. the X_{AxC0R8} and X_{AxC1R8} datasets contain 2458 and 2577 images, respectively, and 3. the $X_{AxC0R24}$ and $X_{AxC1R24}$ datasets contain 2399 and 2646 images, respectively. Each of this test subset underwent an iterative version of the OP attack [25], that targeted the approximate implementations of the CNN, consisting in six adversarial datasets. In specific, we have implemented an incremental version of such an algorithm, i.e., the algorithm searches and modifies an increasing number of pixels. The algorithm is initialized with a number of pixels to be changed equal to one. If no valid solution is found within the maximum number of iterations – changing a pixel does not lead to a successful adversarial image –, the number of pixels to be changed is increased by one. This approach is repeated until the maximum number of pixels to search and modify is reached. Thus, we have set the maximum number of pixels to be changed for each image to 50 and the maximum number of iterations – generation of the DE genetic algorithm – to 51.

Table 4.6: Number of images in the clean and adversarial datasets.

Source AxNN	Clean Dataset (# images)	Adversarial Dataset (# images)
AxC0M	2254	2201
AxC1M	2361	2307
AxC0R8	2458	2309
AxC1R8	2577	2390
AxC0R24	2399	2262
AxC1R24	2646	2381

Table 4.6 reports the number of images in each clean and adversarial datasets, for each source AxNN.

Results

Finally, once generated the adversarial datasets, the image in each of the datasets are transferred to the other networks. Table 4.7 report the transfer-rate of adversarial images from the source AxNNs (in the rows) to all the implementations of the other networks (in the columns).

Table 4.7: Transferability-rate of adversarial attacks, expressed in percentage (%). Rows contain the source AxNN targeted by OP, and from which adversarial attacks are transferred. Columns contain CNNs, with all the implementations, that are target of the transferability.

	FpM	QM	AxC0M	AxC1M	FpR8	QR8	AxC0R8	AxC1R8	FpR24	QR24	AxC0R24	AxC1R24
AxC0M	39.39	55.33	-	57.7	18.62	19.94	19.80	16.90	14.81	13.9	11.67	12.72
AxC1M	46.42	**71.30**	**68.09**	-	18.94	21.15	20.37	17.46	17.29	14.04	11.79	13.35
AxC0R8	32.26	35.81	37.41	35.46	33.21	57.86	-	51.19	25.07	25.07	22.08	24.55
AxC1R8	36.10	39.07	41.12	38.74	39.12	**70.66**	**65.52**	-	28.07	30.58	26.82	30.04
AxC0R24	37.53	41.04	43.22	40.53	34.92	32.87	32.83	29.15	34.75	**79.39**	-	**66.13**
AxC1R24	36.91	39.73	41.53	39.43	34.35	34.14	33.76	34.43	34.14	**65.68**	**71.77**	-

The analysis presented in Table 4.7 clearly demonstrates that an approximate neural network is more effective against approximate and quantized

implementations of a CNN with the same architecture. Specifically, adversarial attacks crafted to fool AxC0M and AxC1M exhibit a transfer-rate against the quantized and approximate versions of MinNet that consistently exceeds 50%. This trend extends to AxC0R8 and AxC1R8, as well as AxC0R24 and AxC1R24.

Furthermore, the smaller the AxNN – in terms of the number of layers – the lower the transferability-rate against deeper CNNs. Indeed, the transferability rate from the approximate MinNet (in both versions) to ResNet8 and ResNet24 consistently remains below 20%, except for the transferability rate from AxC1M to QR8 and AxC0R8, which hovers around $\sim$21%. Similarly, the transferability-rate from the approximate ResNet8 (in both versions) to ResNet24 consistently remains below 30%, except for AxC1R8 to QR24 and AxC0R24, which stands at $\sim$30%. Consequently, successful adversarial sample crafting to deceive MinNet transfers less effectively to ResNet24 compared to when the target is ResNet8.

Conversely, the deeper the source AxNN compared to the target CNN – regardless of the target implementation –, the higher the transferability rate. Notably, the transferability from the approximate versions of ResNet8 to MinNet consistently exceeds 30%, except when the target is FpM and the source is AxC0R8, i.e., the approximate versions with the lowest accuracy (refer to Table 4.3 for details regarding the accuracy). Furthermore, while the transferability rate from ResNet24 to ResNet8 ranges between $\sim$29% and $\sim$35%, the transferability rate from ResNet24 to MinNet ranges between $\sim$37% and $\sim$43%.

Thus, although the transferability is generally poor, it is always possible to transfer successful adversarial samples from an approximate neural network to different implementations of other CNNs. However, the transferability from an AxNN to the quantized and approximate implementations of the same-architecture CNN (almost) away exceeds 65% (bold values in Table 4.7).

4.4.2 Discussion

Although the contribution presented here addressed the Challenge 4, and despite is it possible to transfer adversarial attack from an AxNN to other CNNs, the transferability-rate is generally poor. The transferability rate is even poorer when the architecture of the source AxNN is much smaller – in terms of the number of layers – than the architecture of the target CNN.

To overcome this issue, the main idea may be to taking into account multiple source AxNNs. In specific, the adversarial perturbation to add to the original image may be computed by querying several AxNNs, whose architectures are smaller than the ones of the target CNNs.

4.5 The Role of Approximate Computing on Defense

The last contribution addressees the Challenge 3, since it investigates the effectiveness of input transformation defenses in detecting adversarial attacks against efficient neural networks, with a focus on quantized and approximate implementations [167]. Input transformation has been proposed as a lightweight detection strategy: the key idea is that clean images should remain stable under transformations such as rotation, scaling, or blurring, whereas adversarial images – being artificially crafted to exploit model weaknesses – are expected to induce larger variations in the output predictions. This behavior can be quantified by measuring the divergence between the predictions of the original and transformed inputs, and using it as a signal for adversarial detection. While promising results have been reported for floating-point CNNs, the applicability of this approach to quantized and approximate models has never been explored, which inherently introduce noise into the computation.

To test this detection mechanism, first, we generated QNNs and AxNNs with Inspect-NN, as explained in Section 4.1. Subsequently, we targeted all

the implementation of the CNN with both white-box and black-box attacks, as in Section 4.3. We then applied input transformation, e.g, blurring and rotation, to each input image. For each clean and adversarial image – we denote as x the genic non-transformed image, either the original or the adversarial one – , we applied a digital transformation to obtain x'. Consequently, we computed the error that the network, either the FpNN, the QNN, or the AxNN, make between x and x' as made by Nesti *et al.* in [117]:

$$D(KL(f(x), f(x'))) = \max\{KL(f(x), f(x')), KL(f(x'), f(x))\} \tag{4.7}$$

Here, KL is the Kullback-Leibler (KL)-divergence, f is the classifier, either the original one or the *faulty* ones (quantized and approximates), x is the image (either the clean image or the adversarial counterpart), and x' is the transformed image. To evaluate the detection capability of this method, we compared the statistical distributions of divergences obtained for clean and adversarial datasets across the three implementations of each network. If the distributions are well separated, it is possible to set a threshold τ that allows adversarial inputs to be detected with low error. Conversely, overlapping distributions indicate poor separability, which compromises the effectiveness of the detection.

Figure 4.22 depicts the approach we leveraged to explore the possibility of detecting adversarial attacks against efficient CNNs by leveraging input-transformation and the KL-divergence. We first generate a clean transformed dataset, e.g., a rotated dataset, and an adversarial transformed dataset. Each implementation of the same CNN, that is, the FpNN, the QNN, and the AxNN, is fed with the clean dataset X and the clean transformed datasets X' and, for each pair of images $[x, x']$, the distance $D(KL(f(x), f(x')))$ is computed as described in Equation (4.7), for each classifier. The same operations are performed on the adversarial dataset X^{adv}. Once we computed all the KL-divergences for both the clean and adversarial datasets, and for

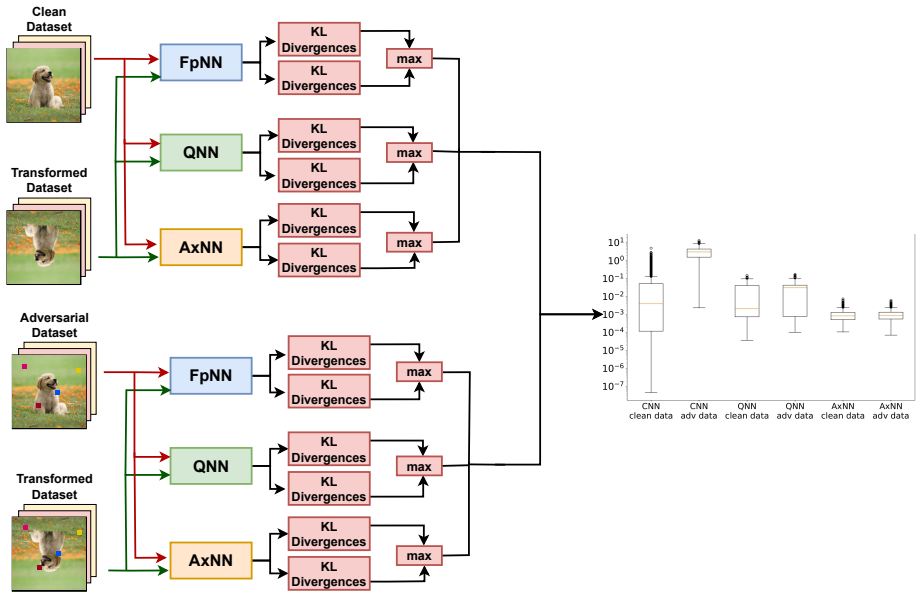


Figure 4.22: The approach we leveraged to explore the possibility of detecting adversarial attacks against efficient implementations of CNNs by leveraging input-transformation and the KL-divergence.

all the implementations of a CNN, our methodology will give a statistical representation of results.

4.5.1 Experimental Evaluation

In order to evaluate the robustness of AxNNs w.r.t. adversarial attacks, we targeted the ResNet networks whose details are depicted in Table 4.3 of section 4.2. Note that, for such a contribution, we considered only the v1 approximate implementation of both ResNet8 and ResNet24. Subsequently we generated adversarial datasets for each implementation of the target CNNs, by following the methodology presented in Section 4.3 – we either attacked directly a CNN implementation or we transferred adversarial images from the

FpNN to the other implementations –. Finally, we computed KL divergences between images (either clean or adversarial) and their transformed versions. The results were statistically analyzed to determine the feasibility of detecting adversarial samples by adopting such a metric in the case of quantized and approximated networks.

Adversarial Attack Generation

In order to generate the adversarial datasets, we targeted the networks with iterative attacks, in both the white and black box scenarios. In specific, we targeted the FpNNs with white-box attacks, i.e., BIM, PGD, and DeepFool. Subsequently, successful adversarial samples are transferred to the QNNs and AxNNs. Note that, in order to design adversarial datasets targeting quantized and approximate version, the FpNN are targeted until the adversarial sample is successfully transferable, or until a maximum number of iterations is reached. For such attacks, we set the maximum number of iterations to 50 in case of BIM and PGD, and to 20 in case of the CW algorithm, while applying an incremental perturbation of $1/255$. For what pertains to the black-box attacks – they allow us to target every implementation of a CNN –, we opted for OP, selecting the size of the population equal to 80, and the number of generations to 97 and 51 for ResNet8 and ResNet24, respectively. For additional information regarding the success rate and transferability rate, readers are directed to consult the experimental evaluation of the first contribution of this Chapter. 4.3.

Input Transformations

For this contribution, we considered two kind of digital transformation, that is, topological and appearance ones. In specific, as topological transformations we selected the (i) *translation*, meaning that images are translated of 2 pixels in both the horizontal and the vertical directions, the (ii) *rotation* to rotate images 45^o in both clockwise and anti-clockwise directions, (iii) *scaling*, by

reducing the image size by 80% in both height and width, and, *flipping*, that mirrors the images. Appearance transformations change the appearance of the image, with no topological changes. We selected (i) the *gaussian blur* applied to filters of size 5×5, and *contrast change*, that scales the intensity of the image on a pixel-by-pixel basis by the value of a parameter $\alpha = 1.5$.

Results

Figures 4.23 and 4.24 depict the statistical behavior of KLs-divergences w.r.t. ResNet8 and ResNet24, respectively. Furthermore, each Subfigure refers to a specific algorithm for generating adversarial examples, while each box-and-whisker plot refers to a specific digital transformation. Specifically, the latter shows the KL divergences between an image and its transformed version, in both the clean and adversarial cases, for the original CNN (the floating-point implementation), the QNN and the AxNN. Finally, for each network, the baseline images are those correctly classified by each implementation of the network itself, i.e., the original floating-point, the quantized and the approximated. Thus, analysis were carried out for 7650 and 7757 images of the CIFAR-10 test-set for ResNet8 and ResNet24, respectively.

As Figures 4.23 and 4.24 show, when targeting the original floating-point networks, both flipping and translation show good detection performance, as well as scaling, for the detection of adversarial samples. since box plots do not overlap, it is possible to easily find a threshold τ above which it is possible to state that an input sample is an adversarial one, with a minimum error. Furthermore, contrast performance is good only in the OP case, while rotation is the worst performing transformation.

However, there are some notable differences when targeting quantized and approximated networks. The detection performances of all input transformations are poor while targeting quantized ResNet8 with both white-box and black-box adversarial generation algorithms, as well as the approximated ResNet8 case. The only exception is represented by the contrast when an-

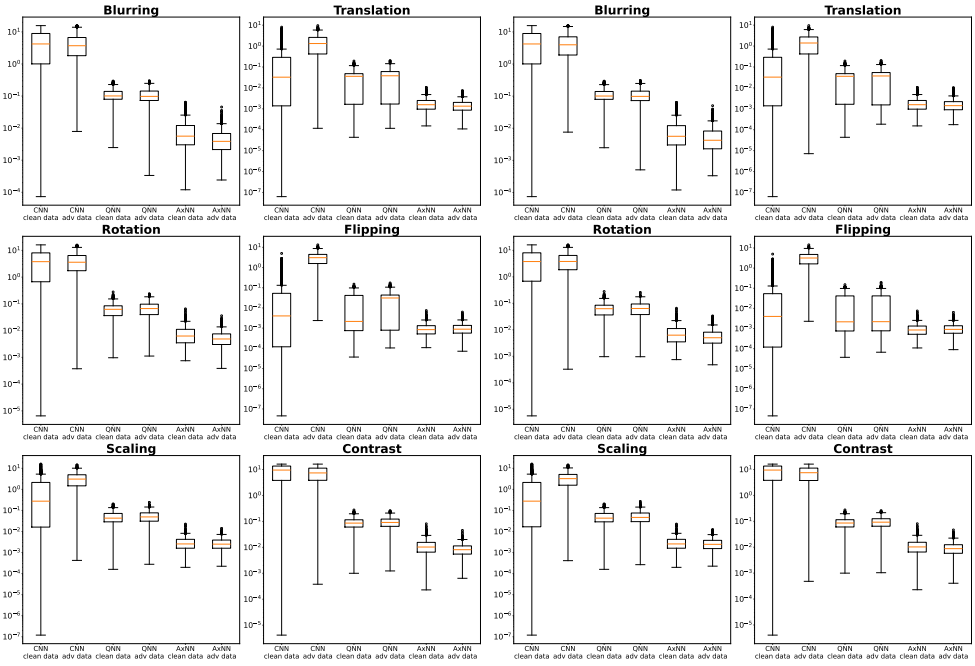
alyzing the approximated ResNet8 targeted by DeepFool: in this case, the performances are good. As for the quantized and approximated ResNet24, scaling performances are good in the case of QNN, while targeted by both white-box and black-box attacks. Finally, contrast performance is good when both quantized and approximated ResNet24 are targeted with DeepFool.

It is worth noting that for all the AxNNs the divergences are statistically smaller than those related to both the original floating-point and the quantized networks. The reason should be that the error introduced by the input-transformation is smaller than the one introduced by the approximated multipliers.

4.5.2 Discussion

Defense methodologies based on input transformations demonstrates high accuracy in detecting adversarial samples when applied to FpNNs, as formerly demonstrated by the scientistic literature. Nevertheless its performance significantly deteriorates when applied to quantized and approximated implementations of the same network. Indeed, this methodology exhibits considerable errors in determining whether an input image is adversarial or not. In specific, flipping, rotation, and scaling are effective techniques for detecting adversarial samples, while, rotation is the least effective transformation when considering FpNNs. Nonetheless, this results do not apply in case of quantization and approximation: almost none of the digital transformation perform well, except for the contrast and the scaling: they slightly show good performances when QNNs and AxNNs are targeted by DeepFool.

Future work will investigates on new open issues emerged while addressing the Challenge 3. Input-transformation based methodologies do not perform well in case of quantized and approximate CNNs. Such a behavior could depend on the fact that the error introduced by the quantization and the approximation *masks* the one introduced by the input-transformations, covering the effects of the latter.



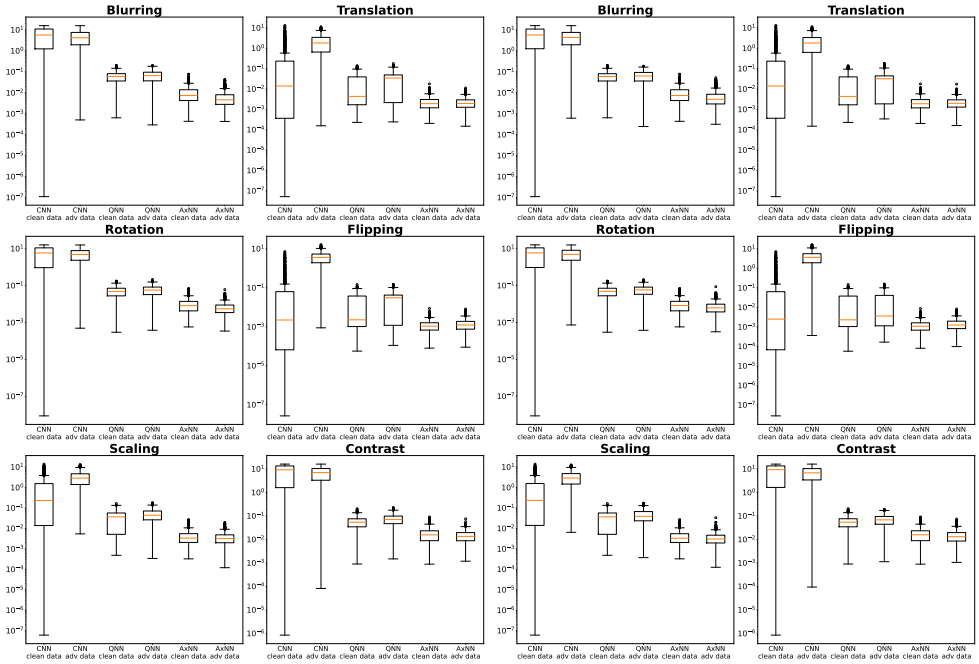
(a) Adversarial examples crafted with BIM.

(b) Adversarial examples crafted with PGD.

(c) Adversarial examples crafted with DeepFool.

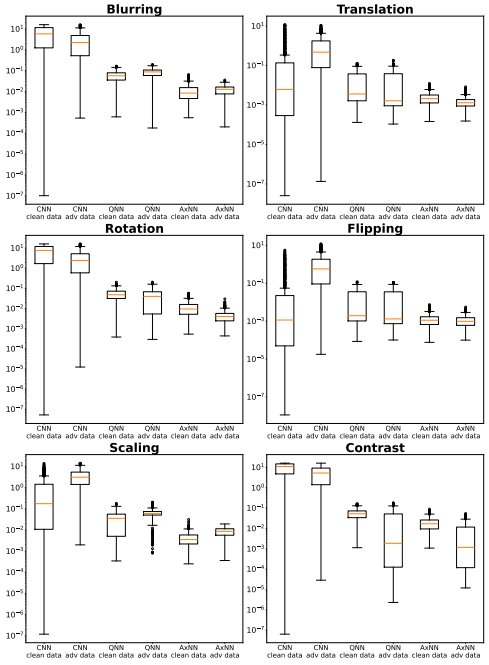
(d) Adversarial examples crafted with OP.

Figure 4.23: KL-divergences between images (either clean or adversarial) and transformed-images while targeting ResNet8.

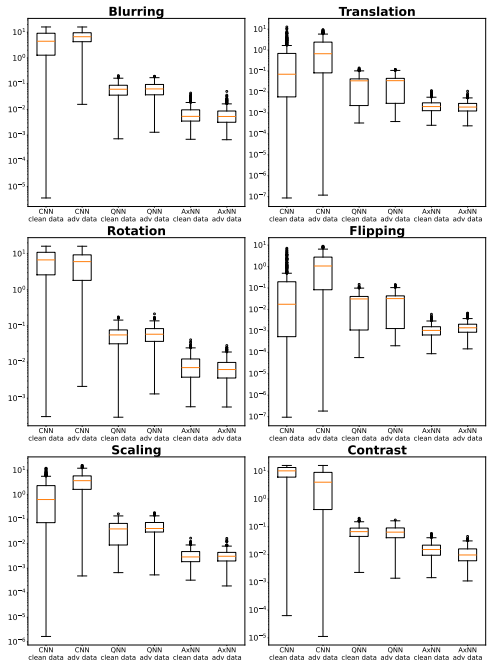


(a) Adversarial examples crafted with BIM.

(b) Adversarial examples crafted with PGD.



(c) Adversarial examples crafted with DeepFool.



(d) Adversarial examples crafted with OP.

Figure 4.24: KL-divergences between images (either clean or adversarial) and transformed-images while targeting ResNet24.

5 Conclusions and Future Works

This thesis has addressed the problem of adversarial robustness in deep learning from two complementary perspectives. The first concerns the design of novel adversarial attack strategies that improve the stealthiness of adversarial examples, making them harder to detect by humans and automated defenses. The second investigates the interaction between efficiency-oriented design methodologies and adversarial robustness, with a particular focus on Approximate Computing (AxC) and quantization. Together, these contributions provide a comprehensive view of how adversarial machine learning evolves both at the algorithmic level (new attack methodologies) and at the architectural level (efficient implementations of Convolutional Neural Networks (CNNs)).

The first contribution of this dissertation was the development of new adversarial attack methodologies aimed at overcoming some of the limitations of existing approaches. Traditional gradient-based attack require non-realistic knowledge regarding the target CNN. Furthermore, black-box query-based methods tend to be inefficient, requiring a large number of interactions with the target model. Finally, both white-boz and black.box attacks often succeed in fooling CNNs but produce perturbations that, although small in norm, are

sometimes visible to the human eye or detectable by preprocessing-based defenses.

To address these limitations, we proposed new attack algorithms designed to maximize stealthiness while maintaining high attack success rates. These contributions include methods that: (i) generate perturbations that remain imperceptible to humans even under low perturbation budgets, (ii) exploit evolutionary or heuristic strategies to improve query efficiency in black-box settings;, and (iii) adaptively tune the perturbation process to balance invisibility and effectiveness.

The experimental results demonstrated that these attacks achieve a better trade-off between stealthiness, efficiency, and effectiveness compared to state-of-the-art methods, highlighting the importance of designing adversarial attacks that are not only theoretically powerful but also practically realizable.

The second major contribution of this thesis focused on the impact of efficiency techniques on adversarial robustness, with a particular emphasis on Approximate Computing. Using the InspectNN framework, we systematically analyzed CNNs in their floating-point, quantized, and approximate implementations. Our investigation covered three dimensions:

- Vulnerabilities of approximate CNNs. We showed that Approximate Neural Networks (AxNNs) are still vulnerable to adversarial examples, but exhibit selective robustness against gradient-based attacks, due to the disruption of the loss surface induced by approximation.

- Transferability of adversarial examples through approximation. We highlighted that AxNNs can act as effective surrogate models, generating adversarial examples with high transferability to other networks, including floating-point and quantized implementations. This reveals a novel attack vector enabled by AxC.

- Ineffectiveness of input transformation defenses. We demonstrated that detection methods based on input transformations, which are effective

on floating-point models, lose much of their discriminative power when applied to quantized and approximate networks, since approximation noise overlaps with adversarial perturbations.

These findings provide the first comprehensive evidence that AxC plays a dual role in adversarial machine learning: on one side, it introduces partial robustness to some classes of attacks, and on the other side, it reduces the effectiveness of certain defenses and facilitates transferability.

Overall contribution By combining these two research directions, the thesis advances the understanding of adversarial machine learning at both the attack design level and the system level. The new attacks proposed demonstrate how adversaries can achieve stealthier and more efficient perturbations, while the analysis of approximate networks clarifies how efficiency-driven design choices affect adversarial robustness. Together, these results emphasize that ensuring the security of deep learning systems requires a holistic approach that jointly considers attack methodologies, defense mechanisms, and efficiency constraints.

5.1 Future Work

Building on the contributions of this thesis, several directions remain open for future exploration:

- Further research should aim at developing attacks that combine invisibility with adaptive transferability, producing perturbations that remain effective across different architectures and implementations while staying undetectable.
- Current defenses fail in the presence of approximate or quantized networks. New detection or mitigation strategies must explicitly model approximation-induced noise, rather than ignoring it.

- The AxC design paradigm should be extended to include robustness as a design objective, not only efficiency. Multi-objective optimization that balances accuracy, power, area, and adversarial robustness could lead to the next generation of secure and efficient CNNs.
- The contributions of this thesis focused on CNNs for image classification. Extending the analysis to other domains, such as speech recognition, natural language processing, or graph learning, would broaden the applicability of the findings.
- Real-world deployments should be tested on FPGA or ASIC platforms implementing approximate multipliers, to assess whether hardware-level constraints amplify or mitigate adversarial effects.

5.2 Closing Remarks

Adversarial machine learning remains one of the greatest challenges to the safe deployment of Artificial Intelligence (AI) systems. This thesis has shown that progress can be made by addressing the problem from multiple angles: by designing new adversarial attacks that are harder to detect, and by investigating how efficiency-driven design choices such as the AxC design paradigm affect adversarial robustness. The insights provided highlight the importance of integrating security and efficiency considerations into the design of future AI systems, ensuring that Deep Learning (DL) can be both practical and trustworthy in real-world scenarios.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[2] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), 2018.

[3] Asharul Islam Khan and Salim Al-Habsi. Machine learning in computer vision. *Procedia Computer Science*, 167:1444–1451, 2020.

[4] Harsh Ahlawat, Naveen Aggarwal, and Deepti Gupta. Automatic speech recognition: A survey of deep learning techniques and approaches. *International Journal of Cognitive Computing in Engineering*, 2025.

[5] KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[6] VA Binson, Sania Thomas, M Subramoniam, J Arun, S Naveen, and S Madhu. A review of machine learning algorithms for biomedical applications. *Annals of Biomedical Engineering*, 52(5):1159–1183, 2024.

[7] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. A survey on adversarial attacks

and defences. *CAAI Transactions on Intelligence Technology*, 6(1): 25–45, 2021. ISSN 2468-2322. doi: 10.1049/cit2.12028. URL https://onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12028. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/cit2.12028.

[8] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems*, 23(6):4909–4926, 2021.

[9] Shervin Minaee, Amirali Abdolrashidi, Hang Su, Mohammed Bennamoun, and David Zhang. Biometrics recognition using deep learning: A survey. *Artificial Intelligence Review*, 56(8):8647–8695, 2023.

[10] Yanjinlkham Myagmar-Ochir and Wooseong Kim. A survey of video surveillance systems in smart city. *Electronics*, 12(17):3567, 2023.

[11] Manuella Kadar and Daniela Onita. A deep cnn for image analytics in automated manufacturing process control. In *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–5, 2019. doi: 10.1109/ECAI46879.2019.9042159.

[12] Sunday Adeola Ajagbe, Matthew O Adigun, Joseph B Awotunde, John B Oladosu, and Yetunde J Oguns. Internet of things enabled convolutional neural networks: applications, techniques, challenges, and prospects. *IoT-enabled Convolutional Neural Networks: Techniques and Applications*, pages 27–63, 2023.

[13] Shuang Cong and Yang Zhou. A review of convolutional neural network architectures and their optimizations. *Artificial Intelligence Review*, 56(3):1905–1969, 2023.

[14] Sheng Xu, Anran Huang, Lei Chen, and Baochang Zhang. Convolutional neural network pruning: A survey. In *2020 39th Chinese Control Conference (CCC)*, pages 7458–7463. IEEE, 2020.

[15] Sridhar Swaminathan, Deepak Garg, Rajkumar Kannan, and Frederic Andres. Sparse low rank factorization for deep neural network compression. *Neurocomputing*, 398:185–196, 2020.

[16] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Trans. Intell. Syst. Technol.*, 14(6),

November 2023. ISSN 2157-6904. doi: 10.1145/3623402. URL https://doi.org/10.1145/3623402.

[17] Mohammad Saeed Ansari, Vojtech Mrazek, Bruce F. Cockburn, Lukas Sekanina, Zdenek Vasicek, and Jie Han. Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(2):317–328, February 2020. ISSN 1557-9999. doi: 10.1109/TVLSI.2019.2940943. URL https://ieeexplore.ieee.org/abstract/document/8863138.

[18] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, February 2014. URL http://arxiv.org/abs/1312.6199. arXiv:1312.6199 [cs].

[19] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

[20] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=BJm4T4Kgx.

[21] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HJGU3Rodl.

[22] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

[23] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep

neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[24] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[25] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[26] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *European conference on computer vision*, pages 484–501. Springer, 2020.

[27] Hafsa Habehh and Suril Gohel. Machine learning in healthcare. *Current genomics*, 22(4):291–300, 2021.

[28] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. *ACM Computing Surveys*, 56(4):1–34, 2023.

[29] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

[30] Yang Lu. Artificial intelligence: a survey on evolution, models, applications and future trends. *Journal of Management Analytics*, 6(1): 1–29, 2019.

[31] Ethem Alpaydin. *Machine learning*. MIT press, 2021.

[32] Jinming Zou, Yi Han, and Sung-Sau So. Overview of artificial neural networks. *Artificial neural networks: methods and applications*, pages 14–22, 2009.

[33] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[34] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[35] Leiyu Chen, Shaobo Li, Qiang Bai, Jing Yang, Sanlong Jiang, and Yanming Miao. Review of image classification algorithms based on convolutional neural networks. *Remote Sensing*, 13(22):4712, 2021.

[36] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[37] Moez Krichen. Convolutional neural networks: A survey. *Computers*, 12(8):151, 2023.

[38] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[39] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S Yu, and Lichao Sun. A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt. *arXiv preprint arXiv:2303.04226*, 2023.

[40] Sahil, Pratham, Neha, Parveen Sadotra, Pradeep Chouksey, Mayank Chopra, and Aditya Thakur. Generative ai for the creation of images. In *International Conference on Information Technology*, pages 193–207. Springer, 2024.

[41] Biswanath Saha. Generative ai for text generation: Advances and applications in natural language processing. *Journal of Computer Allied Intelligence (JCAI, ISSN: 2584-2676)*, 3(1), 2025.

[42] Hao-Wen Dong. *Generative AI for Music and Audio*. University of California, San Diego, 2024.

[43] Arghavan Moradi Dakhel, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Hironori Washizaki. Generative ai for software development: A family of studies on code generation. In *Generative AI for Effective Software Development*, pages 151–172. Springer, 2024.

[44] Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He. Autoencoder and its various variants. In *2018 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 415–419. IEEE, 2018.

[45] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. A review on generative adversarial networks: Algorithms, theory, and applications. *IEEE transactions on knowledge and data engineering*, 35(4):3313–3332, 2021.

[46] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[47] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*, 3, 2023.

[48] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45, 2024.

[49] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[50] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[51] Joana C. Costa, Tiago Roxo, Hugo Proença, and Pedro Ricardo Morais Inácio. How Deep Learning Sees the World: A Survey on Adversarial Attacks & Defenses. *IEEE Access*, 12:61113–61136, 2024. ISSN 2169-3536. doi: 10.1109/ACCESS.2024.3395118. URL

https://ieeexplore.ieee.org/abstract/document/10510296. Conference Name: IEEE Access.

[52] Samer Y. Khamaiseh, Derek Bagagem, Abdullah Al-Alaj, Mathew Mancino, and Hakam W. Alomari. Adversarial Deep Learning: A Survey on Adversarial Attacks and Defense Mechanisms on Image Classification. *IEEE Access*, 10:102266–102291, 2022. ISSN 2169-3536. doi: 10.1109/ACCESS.2022.3208131. Conference Name: IEEE Access.

[53] Shuai Zhou, Chi Liu, Dayong Ye, Tianqing Zhu, Wanlei Zhou, and Philip S Yu. Adversarial attacks and defenses in deep learning: From a perspective of cybersecurity. *ACM Computing Surveys*, 55(8):1–39, 2022.

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[56] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[57] Nicolas Papernot, Patrick McDaniel, Ian 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*, pages 506–519, 2017.

[58] Zhiyi Tian, Lei Cui, Jie Liang, and Shui Yu. A comprehensive survey on poisoning attacks and countermeasures in machine learning. *ACM Comput. Surv.*, 55(8), December 2022. ISSN 0360-0300. doi: 10.1145/3551636. URL https://doi.org/10.1145/3551636.

[59] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018*

IEEE European symposium on security and privacy (EuroS&P), pages 399–414. IEEE, 2018.

[60] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard A. Moser, Alina Oprea, Battista Biggio, Marcello Pelillo, and Fabio Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Comput. Surv.*, 55(13s), July 2023. ISSN 0360-0300. doi: 10.1145/3585385. URL https://doi.org/10.1145/3585385.

[61] Marius Kloft and Pavel Laskov. Online anomaly detection under adversarial impact. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 405–412. JMLR Workshop and Conference Proceedings, 2010.

[62] Battista Biggio, Blaine Nelson, and Pavel Laskov. Support vector machines under adversarial label noise. In *Asian conference on machine learning*, pages 97–112. PMLR, 2011.

[63] Wenbo Jiang, Hongwei Li, Sen Liu, Xizhao Luo, and Rongxing Lu. Poisoning and evasion attacks against deep learning algorithms in autonomous vehicles. *IEEE Transactions on Vehicular Technology*, 69 (4):4439–4449, 2020. doi: 10.1109/TVT.2020.2977378.

[64] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas. A taxonomy and survey of attacks against machine learning. *Computer Science Review*, 34: 100199, 2019.

[65] Zhihao Zheng and Pengyu Hong. Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks. *Advances in neural information processing systems*, 31, 2018.

[66] Hanrui Wang, Shuo Wang, Zhe Jin, Yandan Wang, Cunjian Chen, and Massimo Tistarelli. Similarity-based gray-box adversarial attack against deep face recognition. In *2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021)*, pages 1–8, 2021. doi: 10.1109/FG52635.2021.9667076.

[67] Raz Lapid and Moshe Sipper. I see dead people: Gray-box adversarial attack on image-to-text models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 277–289. Springer, 2023.

[68] Salvatore Della Torca, Valentina Casola, and Simone Izzo. N-pixels: a novel grey-box adversarial attack for fooling convolutional neural networks. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, pages 1539–1547, 2025.

[69] Samer Y Khamaiseh, Izzat Alsmadi, and Abdullah Al-Alaj. Deceiving machine learning-based saturation attack detection systems in sdn. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 44–50, 2020. doi: 10.1109/NFV-SDN50289.2020.9289908.

[70] Ehsan Nowroozi, Yassine Mekdad, Mohammad Hajian Berenjestanaki, Mauro Conti, and Abdeslam El Fergougui. Demystifying the Transferability of Adversarial Attacks in Computer Networks. *IEEE Transactions on Network and Service Management*, 19(3):3387–3400, September 2022. ISSN 1932-4537, 2373-7379. doi: 10.1109/TNSM.2022.3164354. URL https://ieeexplore.ieee.org/document/9747933/.

[71] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[72] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, 2006.

[73] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[74] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11302. URL https://ojs.aaai.org/index.php/AAAI/article/view/11302.

[75] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 03 2005. ISSN 1369-7412. doi: 10.1111/j.1467-9868.2005.00503.x. URL https://doi.org/10.1111/j.1467-9868.2005.00503.x.

[76] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[77] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[78] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[79] Philip Doldo, Derek Everett, Amol Khanna, Andre T Nguyen, and Edward Raff. Stop walking in circles! bailing out early in projected gradient descent. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pages 6373–6382, June 2025.

[80] 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)*, pages 372–387, 2016. doi: 10.1109/EuroSP.2016.36.

[81] Lei Wu and Zhanxing Zhu. Towards understanding and improving the transferability of adversarial examples in deep neural networks. In Sinno Jialin Pan and Masashi Sugiyama, editors, *Proceedings of The 12th Asian Conference on Machine Learning*, volume 129 of *Proceedings of Machine Learning Research*, pages 837–850. PMLR, 18–20 Nov 2020. URL https://proceedings.mlr.press/v129/wu20a.html.

[82] Huy Phan, Yi Xie, Siyu Liao, Jie Chen, and Bo Yuan. Cag: A real-time low-cost enhanced-robustness high-transferability content-aware adversarial attack generator. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5412–5419, Apr. 2020. doi: 10.1609/aaai.v34i04.5990. URL https://ojs.aaai.org/index.php/AAAI/article/view/5990.

[83] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes,

N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

[84] Zhaohui Che, Ali Borji, Guangtao Zhai, Suiyi Ling, Jing Li, and Patrick Le Callet. A new ensemble adversarial attack powered by long-term gradient memories. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3405–3413, Apr. 2020. doi: 10.1609/aaai.v34i04.5743. URL https://ojs.aaai.org/index.php/AAAI/article/view/5743.

[85] Zhibo Wang, Hengchang Guo, Zhifei Zhang, Wenxin Liu, Zhan Qin, and Kui Ren. Feature importance-aware transferable adversarial attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7639–7648, October 2021.

[86] Xinwei Zhang, Tianyuan Zhang, Yitong Zhang, and Shuangcheng Liu. Enhancing the transferability of adversarial attacks with stealth preservation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2915–2925, June 2024.

[87] Kunyu Wang, Xuanran He, Wenxuan Wang, and Xiaosen Wang. Boosting adversarial transferability by block shuffle and rotation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24336–24346, June 2024.

[88] Xiaosen Wang and Kun He. Enhancing the transferability of adversarial attacks through variance tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1924–1933, June 2021.

[89] Zheng Yuan, Jie Zhang, Yunpei Jia, Chuanqi Tan, Tao Xue, and Shiguang Shan. Meta gradient adversarial attack. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7748–7757, October 2021.

[90] Yuyang Long, Qilong Zhang, Boheng Zeng, Lianli Gao, Xianglong Liu, Jian Zhang, and Jingkuan Song. Frequency domain model augmentation for adversarial attack. In *European conference on computer vision*, pages 549–566. Springer, 2022.

[91] Jianping Zhang, Jen-tse Huang, Wenxuan Wang, Yichen Li, Weibin Wu, Xiaosen Wang, Yuxin Su, and Michael R. Lyu. Improving the transferability of adversarial samples by path-augmented method. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8173–8182, June 2023.

[92] Yuhang Zhao, Jun Zheng, Xianfeng Gao, Lu Liu, Yaoyuan Zhang, and Quanxin Zhang. Enhancing the transferability of adversarial attacks via scale enriching. *Neural Networks*, 189:107549, 2025. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2025.107549. URL https://www.sciencedirect.com/science/article/pii/S0893608025004289.

[93] 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*, pages 15–26, 2017.

[94] Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. Simple black-box adversarial attacks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2484–2493. PMLR, 09–15 Jun 2019.

[95] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

[96] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=SyZI0GWCZ.

[97] 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)*, pages 1277–1294, 2020. doi: 10.1109/SP40000.2020.00045.

[98] Panagiotis Bountakas, Apostolis Zarras, Alexios Lekidis, and Christos Xenakis. Defense strategies for adversarial machine learning: A survey. *Computer Science Review*, 49:100573, 2023.

[99] Mengnan Zhao, Lihe Zhang, Jingwen Ye, Huchuan Lu, Baocai Yin, and Xinchao Wang. Adversarial training: A survey, 2024. URL https://arxiv.org/abs/2410.15042.

[100] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvari. Learning with a strong adversary, 2016. URL https://arxiv.org/abs/1511.03034.

[101] Swami Sankaranarayanan, Arpit Jain, Rama Chellappa, and Ser Nam Lim. Regularizing deep networks using efficient layerwise adversarial training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11688. URL https://ojs.aaai.org/index.php/AAAI/article/view/11688.

[102] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/7503cfacd12053d309b6bed5c89de212-Paper.pdf.

[103] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rkZvSe-RZ.

[104] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *Advances in neural information processing systems*, 27, 2014.

[105] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://arxiv.org/abs/1503.02531.

[106] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, 2016. doi: 10.1109/SP.2016.41.

[107] Micah Goldblum, Liam Fowl, Soheil Feizi, and Tom Goldstein. Adversarially robust distillation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3996–4003, 2020.

[108] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Defense against adversarial attacks using high-level representation guided denoiser. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[109] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJUYGxbCW.

[110] Guneet S. Dhillon, Kamyar Azizzadenesheli, Zachary C. Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1uR4GZRZ.

[111] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan L. Yuille. Mitigating adversarial effects through randomization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Sk9yuql0Z.

[112] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=BkJ3ibb0-.

[113] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017. URL https://arxiv.org/abs/1701.07875.

[114] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi N. R. Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E. Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=B1gJ1L2aW.

[115] Yuanyuan Qing, Tao Bai, Zhuotao Liu, Pierre Moulin, and Bihan Wen. Detection of adversarial attacks via disentangling natural images and perturbations. *IEEE Transactions on Information Forensics and Security*, 19:2814–2825, 2024. doi: 10.1109/TIFS.2024.3352837.

[116] Gwonsang Ryu and Daeseon Choi. Detection of adversarial attacks based on differences in image entropy. *International Journal of Information Security*, 23(1):299–314, 2023.

[117] Federico Nesti, Alessandro Biondi, and Giorgio Buttazzo. Detecting adversarial examples by input transformations, defense perturbations, and voting. *IEEE Transactions on Neural Networks and Learning Systems*, 34(3):1329–1341, 2023. doi: 10.1109/TNNLS.2021.3105238.

[118] Min Li, Zihao Huang, Lin Chen, Junxing Ren, Miao Jiang, Fengfa Li, Jitao Fu, and Chenghua Gao. Contemporary advances in neural network quantization: A survey. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2024. doi: 10.1109/IJCNN60899.2024.10650109.

[119] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2018. URL http://jmlr.org/papers/v18/16-456.html.

[120] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[121] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. A survey on deep neural network compression: Challenges, overview, and solutions. *arXiv preprint arXiv:2010.03954*, 2020.

[122] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[123] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgO66VKDS.

[124] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[125] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), March 2016. ISSN 0360-0300. doi: 10.1145/2893356. URL https://doi.org/10.1145/2893356.

[126] Pooja Choudhary, Lava Bhargava, Virendra Singh, and Ashok Kumar Suhag. Approximate computing: Evolutionary methods for functional approximation of digital circuits. *Materials Today: Proceedings*, 66: 3487–3492, 2022.

[127] J.Y.F. Tong, D. Nagle, and R.A. Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3): 273–286, 2000. doi: 10.1109/92.845894.

[128] Georgios Keramidas, Chrysa Kokkala, and Iakovos Stamoulis. Clumsy value cache: An approximate memoization technique for mobile gpu fragment shaders. In *Workshop on approximate computing (WAPCO'15)*, page 6, 2015.

[129] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. Load value approximation. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 127–139, 2014. doi: 10.1109/MICRO.2014.22.

[130] Zhixi Yang, Ajaypat Jain, Jinghang Liang, Jie Han, and Fabrizio Lombardi. Approximate xor/xnor-based adders for inexact computing. In *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*, pages 690–693, 2013. doi: 10.1109/NANO.2013.6720793.

[131] Haider A.F. Almurib, T. Nandha Kumar, and Fabrizio Lombardi. Inexact designs for approximate low power addition by cell replacement. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 660–665, 2016.

[132] Weiqiang Liu, Tian Cao, Peipei Yin, Yuying Zhu, Chenghua Wang, Earl E. Swartzlander, and Fabrizio Lombardi. Design and analysis of approximate redundant binary multipliers. *IEEE Transactions on Computers*, 68(6):804–819, 2019. doi: 10.1109/TC.2018.2890222.

[133] Milan Češka, Jiří Matyaš, Vojtech Mrazek, Lukas Sekanina, Zdenek Vasicek, and Tomas Vojnar. Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 416–423, 2017. doi: 10.1109/ICCAD.2017.8203807.

[134] Salvatore Barone, Marcello Traiola, Mario Barbareschi, and Alberto Bosio. Multi-objective application-driven approximate design method. *IEEE Access*, 9:86975–86993, 2021. doi: 10.1109/ACCESS.2021.3087858.

[135] Syed Shakib Sarwar, Swagath Venkataramani, Aayush Ankit, Anand Raghunathan, and Kaushik Roy. Energy-efficient neural computing with approximate multipliers. *J. Emerg. Technol. Comput. Syst.*, 14 (2), July 2018. ISSN 1550-4832. doi: 10.1145/3097264. URL https://doi.org/10.1145/3097264.

[136] Mohammad Saeed Ansari, Vojtech Mrazek, Bruce F. Cockburn, Lukas Sekanina, Zdenek Vasicek, and Jie Han. Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28 (2):317–328, 2020. doi: 10.1109/TVLSI.2019.2940943.

[137] Syed Shakib Sarwar, Swagath Venkataramani, Aayush Ankit, Anand Raghunathan, and Kaushik Roy. Energy-Efficient Neural Computing with Approximate Multipliers. *J. Emerg. Technol. Comput. Syst.*, 14

(2):16:1–16:23, 2018. ISSN 1550-4832. doi: 10.1145/3097264. URL https://dl.acm.org/doi/10.1145/3097264.

[138] Yulong Yang, Chenhao Lin, Qian Li, Zhengyu Zhao, Haoran Fan, Dawei Zhou, Nannan Wang, Tongliang Liu, and Chao Shen. Quantization aware attack: Enhancing transferable adversarial attacks by model quantization. *IEEE Transactions on Information Forensics and Security*, 19:3265–3278, 2024. doi: 10.1109/TIFS.2024.3360891.

[139] Rémi Bernhard, Pierre-Alain Moellic, and Jean-Max Dutertre. Impact of low-bitwidth quantization on the adversarial robustness for embedded neural networks. In *2019 International Conference on Cyberworlds (CW)*, pages 308–315. IEEE, 2019.

[140] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryetZ20ctX.

[141] Kirsty Duncan, Ekaterina Komendantskaya, Robert Stewart, and Michael Lones. Relative robustness of quantized neural networks against adversarial attacks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. doi: 10.1109/IJCNN48605.2020.9207596.

[142] Micah Gorsline, James Smith, and Cory Merkel. On the adversarial robustness of quantized neural networks. In *Proceedings of the 2021 Great Lakes Symposium on VLSI*, GLSVLSI '21, page 189–194, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383936. doi: 10.1145/3453688.3461755. URL https://doi.org/10.1145/3453688.3461755.

[143] Qun Li, Yuan Meng, Chen Tang, Jiacheng Jiang, and Zhi Wang. Investigating the impact of quantization on adversarial robustness. In *5th Workshop on practical ML for limited/low resource settings*, 2024. URL https://openreview.net/forum?id=TQnw5RIeK2.

[144] Amira Guesmi, Ihsen Alouani, Khaled N. Khasawneh, Mouna Baklouti, Tarek Frikha, Mohamed Abid, and Nael Abu-Ghazaleh. Defensive approximation: securing cnns using approximate computing. In *Proceedings of the 26th ACM International Conference on Architectural*

Support for Programming Languages and Operating Systems, ASPLOS '21, page 990–1003, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446747. URL https://doi.org/10.1145/3445814.3446747.

[145] Mohamed A. Neggaz, Ihsen Alouani, Pablo R. Lorenzo, and Smail Niar. A reliability study on cnns for critical embedded systems. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 476–479, 2018. doi: 10.1109/ICCD.2018.00077.

[146] Mohamed A. Neggaz, Ihsen Alouani, Smail Niar, and Fadi Kurdahi. Are cnns reliable enough for critical applications? an exploratory study. *IEEE Design & Test*, 37(2):76–83, 2020. doi: 10.1109/MDAT.2019.2952336.

[147] Ayesha Siddique and Khaza Anuarul Hoque. Is approximation universally defensive against adversarial attacks in deep neural networks? In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 364–369, 2022. doi: 10.23919/DATE54114.2022.9774563.

[148] Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, and Lukas Sekanina. Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 258–261, 2017. doi: 10.23919/DATE.2017.7926993.

[149] Ehsan Atoofian. Increasing robustness against adversarial attacks through ensemble of approximate multipliers. In *2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–8, 2022. doi: 10.1109/NAS55553.2022.9925476.

[150] Amir Momeni, Jie Han, Paolo Montuschi, and Fabrizio Lombardi. Design and analysis of approximate compressors for multiplication. *IEEE Transactions on Computers*, 64(4):984–994, 2015. doi: 10.1109/TC.2014.2308214.

[151] Sanghyun Hong, Michael-Andrei Panaitescu-Liess, Yigitcan Kaya, and Tudor Dumitras. Qu-anti-zation: Exploiting quantization artifacts for achieving adversarial outcomes. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing*

Systems, volume 34, pages 9303–9316. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/4d8bd3f7351f4fee76ba17594f070ddd-Paper.pdf.

[152] MD Ma and R Fergus. Visualizing and understanding convolutional networks. *Computer Vision–ECCV 2014*, 2013.

[153] Raniah Zaheer and Humera Shaziya. Gpu-based empirical evaluation of activation functions in convolutional neural networks. In *2018 2nd international conference on inventive systems and control (ICISC)*, pages 769–773. IEEE, 2018.

[154] CC Nworu, JE Ekpenyong, J Chisimkwuo, G Okwara, OJ Agwu, and NC Onyeukwu. the effects of modified relu activation functions in image classification. *J Biomed. Eng. Med. Dev.*, 7:237, 2022.

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

[156] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[157] Jeremy Elson, John R Douceur, Jon Howell, and Jared Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. *CCS*, 7:366–374, 2007.

[158] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[159] Ranjita Thapa, Kai Zhang, Noah Snavely, Serge Belongie, and Awais Khan. The plant pathology challenge 2020 data set to classify foliar disease of apples. *Applications in plant sciences*, 8(9):e11390, 2020.

[160] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[161] Horea Mureşan and Mihai Oltean. Fruit recognition from images using deep learning. *Acta Universitatis Sapientiae, Informatica*, 10(1):26–42, August 2018. ISSN 2066-7760. doi: 10.2478/ausi-2018-0002. URL https://acta.sapientia.ro/en/series/informatica/publications/informatica-contents-of-volume-10-number-1-2018/fruit-recognition-from-images-using-deep-learning.

[162] Sven Gowal, Sylvestre-Alvise Rebuffi, Olivia Wiles, Florian Stimberg, Dan Andrei Calian, and Timothy A Mann. Improving robustness using generated data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4218–4233. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/21ca6d0cf2f25c4dbb35d8dc0b679c3f-Paper.pdf.

[163] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2206–2216. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/croce20b.html.

[164] Jary Pomponi, Simone Scardapane, and Aurelio Uncini. Pixle: a fast and effective black-box attack based on rearranging pixels. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, July 2022. doi: 10.1109/IJCNN55064.2022.9892966. URL https://ieeexplore.ieee.org/document/9892966/?arnumber=9892966. ISSN: 2161-4407.

[165] Mario Barbareschi, Salvatore Barone, Valentina Casola, and Salvatore Della Torca. Error resiliency and adversarial robustness in convolutional neural networks: An empirical analysis. In *IFIP International Internet of Things Conference*, pages 149–160. Springer, 2024.

[166] Valentina Casola and Salvatore Della Torca. Transfer adversarial attacks through approximate computing. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ARES '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400717185. doi: 10.1145/3664476.3670449. URL https://doi.org/10.1145/3664476.3670449.

[167] Salvatore Barone, Valentina Casola, and Salvatore Della Torca. Ineffectiveness of digital transformations for detecting adversarial attacks

against quantized and approximate cnns. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 290–295, 2024. doi: 10.1109/CSR61664.2024.10679345.

[168] Mohammad Hasan Ahmadilivani, Mario Barbareschi, Salvatore Barone, Alberto Bosio, Masoud Daneshtalab, Salvatore Della Torca, Gabriele Gavarini, Maksim Jenihhin, Jaan Raik, Annachiara Ruospo, Ernesto Sanchez, and Mahdi Taheri. Special session: Approximation and fault resiliency of dnn accelerators. In *2023 IEEE 41st VLSI Test Symposium (VTS)*, pages 1–10, 2023. doi: 10.1109/VTS56346.2023.10140043.

[169] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. arxiv 2021. *arXiv preprint arXiv:2106.08295*, 4, 2021.

[170] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

[171] 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 *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[172] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. Tensorflow lite micro: Embedded machine learning for tinyml systems. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 800–811, 2021. URL https://proceedings.mlsys.org/paper_files/paper/2021/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf.

[173] Vojtech Mrazek, Zdenek Vasicek, Lukas Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. Alwann: Automatic layer-wise approximation of deep neural network accelerators without retraining. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019. doi: 10.1109/ICCAD45719.2019.8942068.

[174] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, Patrick McDaniel, et al. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 10, 2016.

[175] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, et al. Adversarial robustness toolbox v1. 0.0. *arXiv preprint arXiv:1807.01069*, 2018.

[176] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010. doi: 10.1109/ICPR.2010.579.

[177] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

List of figures

2.1 The basic model of a neuron within a neural network. 13

2.2 An example of Convolutional Neural Network (CNN) trained to recognize handwritten digits from the MNIST dataset [36]. Please refer to [7] for further details related to the Figure. . . 15

2.3 Example of an adversarial image from the Reference [19]. The authors apply a perturbation to a panda image from the ImageNet dataset [55], resulting in an adversarial image that is visually indistinguishable from the original one. However, although the target Deep Neural Network (DNN) GoogleLeNet [54] correctly classifies the original image, it produces an incorrect output-label while classifying the adversarial image. 19

2.4 A generic and simple self-driving car example to illustrate a ML system as a data processing pipeline. 22

3.1 Representation of input and output volumes I_z and G_z, respectively, of a feature layers l_z. In this context, f* is the most important fiber within G_z, that corresponds to the most important section S_z^* of the input volume. 60

3.2 The process of back-mapping. The most important section S_z^* within the volume I_z, identified through the most important fiber f* in the volume G_z, corresponds to a set of fibers within the output volume G_{z-1} of the layer l_{z-1}. These fibers are back-mapped in order to identify the section S_{z-1}^* within the input volume I_{z-1}. The entire process is repeated up to the layer l_a 62

3.3 Distribution of the Peak Signal-to-Noise Ratio (PSNR) while targeting ResNet24 (**a**), ResNet50 (**b**), and MobileNet-v3 (**c**) on the CIFAR-10, Dogs vs. Cats, and FGCV7 datasets, respectively. 71

3.4 Distribution of the PSNR while targeting ResNet24 (**a**), ResNet50 (**b**), and MobileNet-v3 ((**c**) MobileNet-v3.) on the CIFAR-10, Dogs vs. Cats, and FGCV7 datasets, respectively. In red, there are additive results obtained by relaxing the *similarity constraint*, with a minimal PSNR to 20 dB. 72

3.5 Example of how N-Pixels (NPs) craft an adversarial image $\hat{x}$ from the original one x in the case of binary classification problem **(a)**, and a multiclass classification problem **(b)**. . . 73

3.6 Distribution of the PSNRs while targeting ResNet24 **(a)**, ResNet50 **(b)**, and MobileNet-v3 **(c)** with Basic Iterative Method (BIM), Projected Gradient Descent (PGD), NPs, One-Pixel (OP), and Simple Black-Box Adversarial Attack (SimBA). 75

3.7 Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by Incremental-Pixels (IPs) while targeting ResNet-24 on the CIFAR-10 test dataset. 88
3.8 Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPs while targeting ResNet-50C on the CIFAR-100 test dataset. . 88
3.9 Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPs while targeting ResNet-50D on the Dogs vs. Cats test dataset. 89
3.10 Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPswhile targeting MobileNet on the FGCV7 test dataset. . . 89
3.11 Distributions of L_0 norm **(a)**, L_∞ norm **(b)**, PSNR **(c)**, and number of queries **(d)** of adversarial images generated by IPs while targeting VGG-16 on the Fruit-360 test dataset. 90

4.1 Methodology for approximating the CNNs. 100
4.2 Lookup-based multiplication 103
4.3 The methodology we developed to evaluate the robustness of AxNNs against adversarial attacks. 109
4.4 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by BIM to fool the floating-point, quantized and approximate implementations of MinNet. 115
4.5 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by PGD to fool the floating-point, quantized and approximate implementations of MinNet. 115
4.6 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by Carlini-Wagner (CW) to fool the floating-point, quantized and approximate implementations of MinNet. 116

4.7 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by DeepFool to fool the floating-point, quantized and approximate implementations of MinNet. 116
4.8 Number of queries **(a)**, and PSNR **(b)**, required by OP to fool the floating-point, quantized and approximate implementations of MinNet. 117
4.9 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by BIM to fool the floating-point, quantized and approximate implementations of ResNet8. 117
4.10 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by PGD to fool the floating-point, quantized and approximate implementations of ResNet8. 118
4.11 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by CW to fool the floating-point, quantized and approximate implementations of ResNet8. 118
4.12 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by DeepFool to fool the floating-point, quantized and approximate implementations of ResNet8. 119
4.13 Number of queries **(a)**, and PSNR **(b)**, required by OP to fool the floating-point, quantized and approximate implementations of ResNet8. 119
4.14 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by BIM to fool the floating-point, quantized and approximate implementations of ResNet24. 120
4.15 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by PGD to fool the floating-point, quantized and approximate implementations of ResNet24. 120
4.16 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by CW to fool the floating-point, quantized and approximate implementations of ResNet24. 121

4.17 Perturbation strength **(a)**, number of queries **(b)**, and PSNR **(c)**, required by DeepFool to fool the floating-point, quantized and approximate implementations of ResNet24. 121
4.18 Number of queries **(a)**, and PSNR **(b)**, required by OP to fool the floating-point, quantized and approximate implementations of ResNet24. 122
4.19 Overview of our methodology – Generating the quantized and approximate implementations. 123
4.20 Overview of our methodology – Crafting the adversarial dataset. 124
4.21 Overview of our methodology – Transferring the adversarial dataset. 125
4.22 The approach we leveraged to explore the possibility of detecting adversarial attacks against efficient implementations of CNNs by leveraging input-transformation and the Kullback-Leibler (KL)-divergence. 131
4.23 KL-divergences between images (either clean or adversarial) and transformed-images while targeting ResNet8. 135
4.24 KL-divergences between images (either clean or adversarial) and transformed-images while targeting ResNet24. 136

List of tables

2.1 List of symbols and the corresponding definitions related to Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs). 17
2.2 List of symbols and the corresponding definitions related to adversarial samples. 21

3.1 Success-rate of ResNet24, ResNet50, and MobileNet-v3 on the CIFAR-10, Dogs vs. Cats, and FGCV7 test datasets, respectively. 68
3.2 Success-rate of N-Pixels (NPs) against ResNet24, Resnet50, and MobileNet-v3. 70
3.3 Success-rate of NPs against ResNet24, Resnet50, and MobileNet-v3. In this case, the *similarity constraint* was relaxed, by setting a minimal Peak Signal-to-Noise Ratio (PSNR) to 20 dB. 72
3.4 Success-rate of NPs, One-Pixel (OP), Simple Black-Box Adversarial Attack (SimBA), Basic Iterative Method (BIM), and Projected Gradient Descent (PGD) against ResNet 24, Resnet50, and Mobilenet-v3. 74

3.5 Accuracy of ResNet-24, ResNet-50C, ResNet-50D, MobileNet-v3, and VGG-16 on the CIFAR-10, CIFAR-100, Dogs vs. Cats, FGCV7, and Fruits-360 test datasets, respectively. . . . 86

3.6 Success-rate of Incremental-Pixels (IPs) against ResNet-24, ResNet-50C, ResNet-50D, MobileNet-v3, and VGG-16 on their respective datasets. 87

3.7 Results obtained while attacking ResNet-24, ResNet-50C, ResNet-50D, MobileNet-v3, and VGG-16 on their respective datasets, with the proposed attack IPs, OP while targeting a single pixel (OP1), OP while targeting 1% of the pixels (OP1%), Square Attack (SQ), and Pixle (PX). For each score, we show the mean and the variance, if present, calculated over all the images on that test dataset. 92

4.1 Error characterization and hardware requirements for approximate circuits taken from the EvoApproxLib-Lite library, as reported in [148]. 105

4.2 Architectural details of the MinNet CNN. 106

4.3 Accuracy, and estimated resource requirements for MinNet, ResNet8 and ResNet24 implementations. 107

4.4 Success rate of BIM, PGD, Carlini-Wagner (CW), DeepFool, and OP against the implementations of MinNet, ResNet8, and ResNet24. Note that, since white-box attacks require the differentiability of loss-function, they are not applicable to the quantized and approximate implementations. 113

4.5 Transferability rate from the floating-point implementation to the quantized and approximate ones, for MinNet, ResNet8, and ResNet24 of BIM, PGD, CW, DeepFool, and OP. 113

4.6 Number of images in the clean and adversarial datasets. . . . 127

4.7 Transferability-rate of adversarial attacks, expressed in percentage (%). Rows contain the source Approximate Neural Network (AxNN) targeted by OP, and from which adversarial attacks are transferred. Columns contain CNNs, with all the implementations, that are target of the transferability. 127