POLITECNICO DI TORINO

Department of Electronics and Telecommunications
Master's Degree in Electronic Engineering

Master's Thesis

Black-Box Adversarial Attacks for Deep Neural Networks and Spiking Neural Networks

Supervisors:
Prof. Maurizio Martina
Prof. Muhammad Shafique
Project Ass. Alberto Marchisio

Candidate:
Giorgio Nanfa

April 3, 2019

Acknowledgments

I am very grateful to everyone helped me during my work of thesis. I want to thank Professor Maurizio Martina, who allowed me to know many aspects of electronics and always gave me a great support during my years spent in Politecnico di Torino. He gave me the possibility to develop my thesis at TU Wien, thanks to the availability and kindness of Professor Muhammad Shafique. I admire him for his dedication to work and the incredible wish to share his wide knowledges and at the same time learn from the students. I consider myself lucky to have worked with two Professors like them. Specially I want to thank my friend and tutor Alberto: he was my main reference in Wien and it was an honour to collaborate with him. He gave me the opportunity to discover a new reality, to know many friends and to live fantastic experiences, beyond every expectation. He always helped me in my work and taught me many important things. Words will never be enough to thank him for his support. I want to thank the group of PhD students who have always been kind and present for opinions and suggestions. I also want to thank Andrea and Ymer: they taught me the importance of working in team and we spent together a lot of moments that i will never forget. My experience in Turin was fantastic thanks to them and many, many people who made these years unforgettable. I want to thank my friend Luigi, since, without his willpower, I will never attend this university and know so many people immediately after my diploma. I will be always grateful to him. Moreover, a great thank to my friends Sandro and Federica, who always gave me their support and taught me the importance of the true friendship, and to my roommate Matteo, very funny and kind person. A particular thank to my High School Professor Giacomo Principato: he believes in a true and honest teacher-student relationship and for me he represents a very important point of reference. I want to thank all my friends, in particular the ones I knew during four years at Collegio Artigianelli: I got experience thanks to them, the community life and all the moments we spent together. The most important thanks are for my family. My parents always supported me in my choices and taught me everything: it is difficult to find the right words to thank them for everything they did for me. A great thank to my sister, my brother, my grandparents and the rest of the family who always believed in me.

Summary

Recently, many adversarial examples have emerged for Deep Neural Networks (DNNs) causing misclassifications. These perturbations, added to the test inputs, are small and imperceptible to humans, but fool the network to mis-predict. However, it is important to evaluate the vulnerabilities of Neural Networks to adversarial noise/examples before such networks can be deployed in safety-critical applications, e.g., autonomous driving, privacy and banking applications and smart healthcare. For this reason, we develop a novel black-box attack methodology to automatically generate targeted imperceptible and robust adversarial examples through a greedy algorithm. The attacks are applied under black-box assumption when the attacker does not know the architecture, the training data and the parameters of the network. Moreover, an attack is targeted when the target class, i.e., the class in which the attacker wants to classify the example, is defined a-priori by the intruder.
We propose two different analyses:

1. We study the vulnerabilities in Spiking Neural Networks (SNNs), i.e. the 3rd generation NNs, applying our novel methodology to the Modified National Institute of Standards and Technology (MNIST) database for:
   a) Spiking Deep Belief Networks (SDBNs)
   b) Liquid State Machines (LSMs)

2. We study the vulnerabilities in Capsule Networks to adversarial attacks, applying our novel methodology and some affine transformation to the German Traffic Sign Recognition Benchmark (GTSRB).

SNN under Attack: are Spiking Deep Belief Networks vulnerable to Adversarial Examples?
Spiking Neural Networks (SNNs) are the third generation of neural network models, and are rapidly emerging as a better design option compared to DNNs, due to their inherent model structure and properties matching the closest to today's understanding of a brain's functionality. They are computationally more powerful than several other NN models, high energy efficient and biologically plausible.
We aim at generating, for the first time, imperceptible and robust adversarial examples for SNNs. The scope of our attack is to generate adversarial images, which

are difficult to be detected by human eyes (imperceptible) and resistant to physical transformations (robust). For the evaluation, as a case study we apply these attacks to a Spiking Deep Belief Network (SDBN) and a DNN having the same number of layers and neurons, to obtain a fair comparison. Deep Belief Networks (DBNs) are multi-layer networks that are widely used for classification problems and implemented in many areas such as visual processing, audio processing, images and text recognition with optimal results. SDBNs improve the energy efficiency and computation speed, as compared to DBNs.
We investigate the vulnerability of SDBNs to random noise and adversarial attacks applied to the MNIST dataset, aiming at identifying the similarities and the differences with respect to DNNs. Our experiments show that, when applying a random noise to a given SDBN, its classification accuracy decreases, by increasing the noise magnitude. Moreover, applying our attack to SDBNs, we observe that, in contrast as the case of DNNs, the output probabilities follow a different behavior: while the adversarial image remains imperceptible, the misclassification is not always guaranteed.

Vulnerability of LSMs to imperceptible and robust adversarial examples
In many applications in which SNNs are used, e.g., tracking systems, decision making and action selection, it is fundamental to take into account not only the input data, but also the temporal information. In order to process this kind of information, it has been demonstrated that recurrent connections in Neural Networks enable to deal with dynamic temporal patterns showing high computational capabilities. Recurrent Neural Networks (RNNs) are dynamical systems characterized by feedback connections that can model reciprocal interactions between the neurons. Despite their great potential, these networks are difficult to train: their training results to be computationally expensive and slow. In order to overcome these difficulties and exploit the advantages of RNNs, it has been studied a method of computing called Liquid State Machine (LSM), a method related to the Reservoir Computing (RC) model. This model is composed of a Reservoir, the fixed recurrent structure that uses spiking neuron models, and a set of readouts, i.e., output neurons. Since these networks play a key role in many important safety-critical applications, their security represents a fundamental topic towards the future of the Machine Learning (ML) in a lot of applications in real life. We consider as case of study a LSM, performing the classification of the MNIST dataset. The input pixels of each image are converted into Poisson spike trains.
We show that, in the most of the cases, the Poisson encoding process limits the performances of our black-box methodology. From one side, in the examples having the starting probabilities not so far from each other, this process represents a weakness, because it results to be easy to fool the network. In this cases we create imperceptible examples, but their robustness is not guaranteed. From the other

side, when the starting gap is high enough, it represents a sort of antidote against imperceptible modifications of the pixels.

CapsAttacks: Robust and Imperceptible Adversarial Attacks on Capsule Networks

Convolutional Neural Networks (CNNs) are specialized to identify and recognize the presence of an object as a feature, without taking into account the spatial relationships across multiple features. Recently CapsuleNets, a specialized Neural Network architecture composed of capsules, have been proposed. CapsuleNets envision an innovative point of view about the representation of the objects in the brain and preserve the hierarchical spatial relationships between them, unlike the CNNs. In CapsuleNets the feature representations are stored inside the capsules in a vector form, in contrast to the scalar form used by the neurons in traditional Neural Networks. The capsules operate in a way similar to the one performed by the artificial neurons and, thanks to the use of vectors instead of scalars, CapsuleNets can obtain a good accuracy using less training data with respect to the CNNs. Recent researches about CapsuleNet architecture and training algorithms have shown competitive results, in terms of accuracy, for image classification task, compared to other state-of-the-art classifiers. Hence, it becomes fundamental to study if and how the CapsuleNets are vulnerable to adversarial examples.

First of all, we investigate the impact of universal attacks on CapsuleNet with different (additive and subtractive) perturbations varying in their magnitudes. Our analyses show that, when the noise is subtracted from the intensity of the pixels, the accuracy of the network decreases quickly when compared to the other case. Moreover, we develop an algorithm to generate black-box targeted imperceptible and robust attacks for the German Traffic Sign Recognition Benchmark, which is more crucial for autonomous vehicle related use cases. We apply the same type of attacks to a 9-layer VGGNet having a similar starting accuracy, compared to the CapsuleNet. Our analyses show that CapsuleNets are more vulnerable than CNNs: CapsuleNet has a much higher learning capability, compared to the VGGNet, but this phenomena has a negative drawback for the machine learning security point of view. Since the VGGNet is deeper and contains a larger number of weights, while the CapsNet can achieve a similar accuracy with a smaller footprint, we observe a disparity in the prediction confidence between the two networks. Hence, we study the behavior of these two networks under some affine transformation to the input images. Also in this case, the VGGNet results to be more robust than CapsuleNet. These results are consistent with the behavior of CapsuleNet observed applying our attack. In an era in which the autonomous driving community is looking for high security of automatic systems in safety-critical environments, the CapsuleNet does not guarantee a sufficient robustness. Hence, further modifications of the CapsuleNet architecture need to be designed to reduce its vulnerability to adversarial attacks.

Table of contents

Chapter 1

Introduction

Recently, many adversarial examples have emerged for Deep Neural Networks (DNNs) causing misclassifications. These perturbations, added to the test inputs, are small and imperceptible to humans, but fool the network to mis-predict. However, it is important to evaluate the vulnerabilities of Neural Networks to adversarial noise/examples before such networks can be deployed in safety-critical applications, e.g., autonomous driving and smart healthcare. In our work we develop a novel black box attack methodology to automatically generate targeted imperceptible and robust adversarial examples through a greedy algorithm. We propose two different analyses:

1. We study the vulnerabilities in Spiking Neural Networks (SNNs), i.e. the 3rd generation NNs, applying our novel methodology to the Modified National Institute of Standards and Technology (MNIST) database.

2. We study the vulnerabilities in Capsule Networks to adversarial attacks, applying our novel methodology to the German Traffic Sign Recognition Benchmark (GTSRB).

We show that our adversarial examples, characterized by the imperceptibility and the robustness, fool the analyzed NNs in different ways, according to their features. Moreover we compare our results applying our crafted inputs to other DNNs and considering other methods generating attacks.

1.1 Overview

The thesis is organized in the following chapters:

- **Chapter 2**: we explain the basic notions of Neural Networks and the knowledge relative to Adversarial Attacks, CapsuleNets and SNNs

- **Chapter 3**: we present a new methodology to automatically create adversarial examples and we test the vulnerability of a SDBN under our attacks and random noise applied to the inputs
- **Chapter 4**: we analyze the vulnerability of Liquid State Machines (LSMs) under our black-box attack
- **Chapter 5**: we analyze the robustness of CapsuleNetwork, under our modified attacks methodology, under some affine transformation and also under random noise applied to the inputs
- **Chapter 6**: we draw the conclusions of our work

Chapter 2

Background

We introduce here the basic concepts relative to the arguments discussed in the thesis. We start explaining the meaning of Artificial Intelligence (AI) and the topics related to it in order to better understand the evolution of the Neural Networks (NNs). Furthermore we explore the topic of Adversarial Attacks and we highlight why they are so dangerous in a lot of applications.

2.1 From Artificial Intelligence to Deep Learning

Artificial intelligence is the science of creating intelligent machines that are capable to achieve specific goals and tasks like humans do. It is a very large topic, going from video games to autonomous driving and including every application in which a machine can learn or predict something. A subset of AI is **Machine Learning** (ML): it indicates the ability of a machine to learn informations (*training*) and solve problems without being explicitly programmed every time [1].

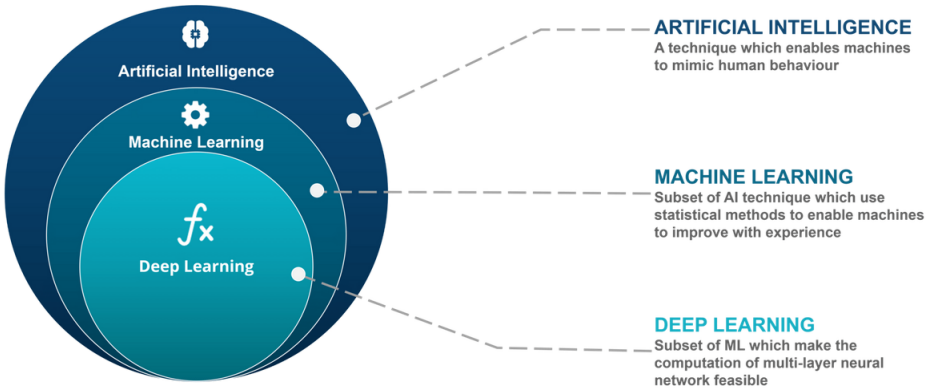


Figure 2.1: Overview of AI set [Source: edureka.co]

Since the humans aim at creating machines that work like the brain, it is possible to talk about *brain inspired computation*. The artificial structures that model the real biological Neural Networks are called **Artificial Neural Networks (ANNs)** or just Neural Networks. They are computational models composed of many layers of artificial neurons: the neuron is the main computational unit of the brain. The structure of a neuron is shown in fig. 2.2.

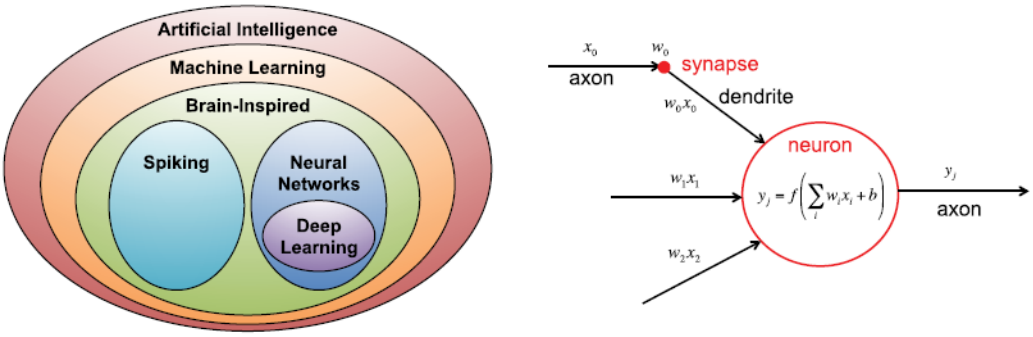


Figure 2.2: Another overview of AI set and the model of an artificial neuron [Source: [1]]

The neurons are connected together in a NN, so the output of one neuron, called *axon*, represents one of the inputs of another one, called *dendrites*. The output of a neuron corresponds to the weighted sum of the inputs. The activation function, indicated as *f*, can be linear or nonlinear. The connection between an axon and a dendrite is called *synapse*: this link between output and input of two neurons (called *presynaptic* and *postsynaptic* neurons respectively) scales the axon output signal by a quantity called *weight*. A NN learns informations updating the values of the weights in response to input stimuli: this process is called *learning or training*. The learning can be:

- **Supervised**: all the training data are labeled (each sample has a correct output, called *label*).
- **Unsupervised**: data are unlabeled, so the NN tries to extract some features from them in order to learn informations.
- **Semi-supervised**: a part of the data is labeled, the other one is unlabeled.
- **Reinforcement**: it is based on a particular reward system, according to the correct or wrong prediction of the NN.

Once the NN is trained using the training data, the values of the weights are determined and the performances of the NN are evaluated on the test data. Hence,

considering a complete dataset, we can distinguish the training data, used for the training process, and the test data, used for the *inference* process. As we can observe in fig. 2.2, **Deep Learning** is a subset of NNs: it contains the NNs, called **Deep Neural Networks (DNNs)**, with more than three layers [1]. In general the first layer is called *input layer*, while the last one *output layer*. The layers between these two ones are called *hidden layers*: the Network is deeper increasing the number of hidden layers. Sze *et al.* [1] clearly describes why DNNs are used in a lot of tasks.
DNNs are capable of learning high-level features with more complexity and abstraction than shallower neural networks. An example that demonstrates this point is using DNNs to process visual data. In these applications, pixels of an image are fed into the first layer of a DNN, and the outputs of that layer can be interpreted as representing the presence of different low-level features in the image, such as lines and edges. At subsequent layers, these features are then combined into a measure of the likely presence of higher level features, e.g., lines are combined into shapes, which are further combined into sets of shapes. And finally, given all this information, the network provides a probability that these high-level features comprise a particular object or scene. This deep feature hierarchy enables DNNs to achieve superior performance in many tasks.

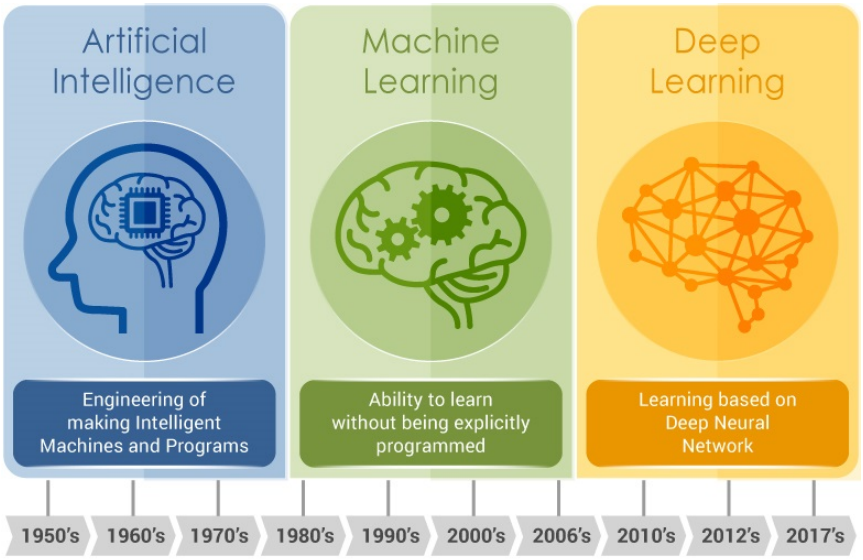


Figure 2.3: Steps from AI to Deep Learning [Source: Embedded Vision Alliance]

2.2 Spiking Neural Networks

2.2.1 Basic knowledge

Spiking Neural Networks (SNNs) are the third generation of neural network models [10], and are rapidly emerging as a better design option compared to DNNs, due to their inherent model structure and properties matching the closest to today's understanding of a brain's functionality. As a result, SNNs result in:

- **Computationally more Powerful than several other NN Models:** a lower number of neurons is required to realize the same computations.
- **High Energy Efficiency:** spiking neurons process the information only when a new spike arrives, so they have lower energy consumption because the spike events are sparse in time [12].
- **Biologically Plausible**: spiking neurons are very similar to the biological ones because they use discrete spikes to compute and transmit information. Biological neurons communicate by generating (*firing*) and propagating spikes. For this reason, SNNs are also highly sensitive to the temporal characteristics of processed data [9] [11].

The spikes, also called *action potentials*, are short electrical pulses that we can observe placing an electrod near to the axon of a neuron: the neurons send these signals across the synapses. Each neuron generates a series of spikes, called **spike train**. The neural information is carried by the timing of the spikes, not by their shapes.
After a neuron generates a spike, it exists a period, called **absolute refractory period**, in which it is impossible that the neuron generates another spike. The duration of this period is about 10 ms. *The effect of a spike on the postsynaptic neuron can be recorded with an intracellular electrode which measures the potential difference $u(t)$ between the interior of the cell and its surroundings. This potential difference is called the* ***membrane potential****. Without any spike input, the neuron is at rest corresponding to a constant membrane potential. After the arrival of a spike, the potential changes and finally decays back to the resting potential. If the change is positive, the synapse is said to be* ***excitatory****. If the change is negative, the synapse is* ***inhibitory****. At rest, the cell membrane has already a strong negative polarization of about -65 mV.* [9]
Hence, considering a presynaptic neuron j and a postsynaptic neuron i, before spikes coming, $u_i(t) = u_{rest}$, where u_{rest} is the membrane potential at rest. When the neuron j fires, the neuron i generates an excitatory or inhibitory postsynaptic potential (**EPSP** or **IPSP**). Since the considered neuron i has got many inputs (dendrites), its membrane potential $u_i(t)$ depends on the spikes coming from each neuron because

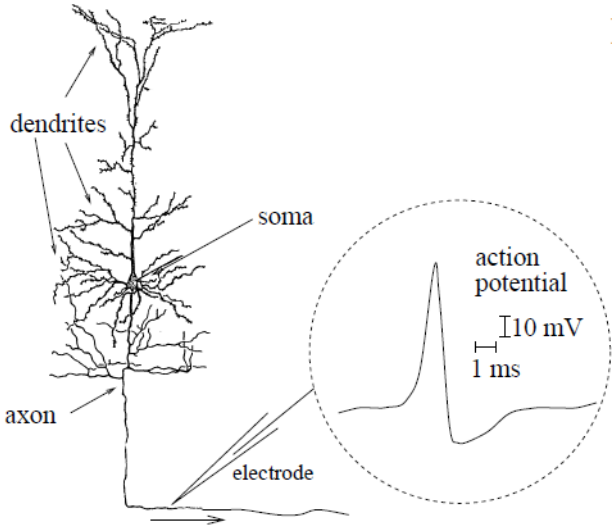


Figure 2.4: Model of a spiking neuron. The duration of the pulse is 1-2 ms, the amplitude is about 100 mV [Source: [9]]

each spike causes a postsynaptic potential, as shown in fig. 2.5. If the sum of all the coming PSPs overcomes a threshold θ, the neuron i fires. Gerstner and Kistler [9] defined the membrane potential of the neuron i as:

$$u_i(t) = \sum_j \sum_f \epsilon_{ij}(t - t_j^{(f)}) + u_{rest}$$

j indicates the index of the presynaptic neuron, t^f the firing time (so the value of time such that $u(t^{(f)}) = \theta$) and $\epsilon_{i,j}$ the PSP. After the spike, the membrane potential does not return directly to the rest value u_{rest}, but it decays slowly. This phenomena is called **spike-afterpotential**: it represents a form of hyperpolarization, i.e., the negative polarization of the membrane increases.

2.2.2 Neuron Models

We briefly analyze two spiking neuron models and finally we compare some models from two points of view: their biological plausibility and their computational cost.

Integrate and Fire
It is the most used model of biological threshold neurons because it is very simple to implement. We can observe its structure [9] in fig. 2.6.
Since the model is composed of a RC circuit, it is not so biologically plausible. We consider the neurons j (presynaptic) and i (postsynaptic). When j fires, a spike goes through the axon and then through a low pass filter. The output is the current

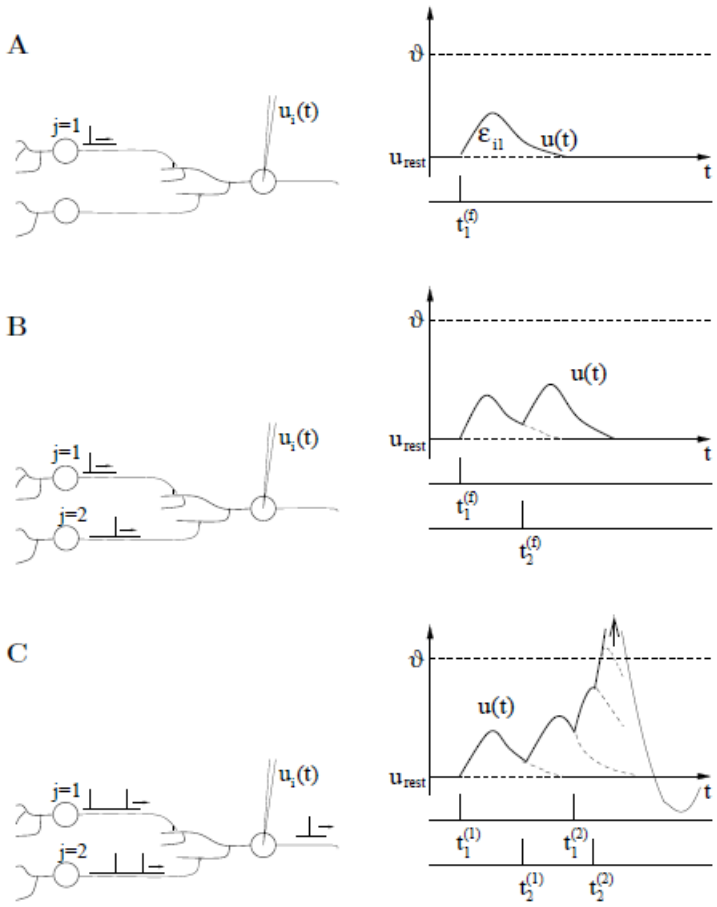


Figure 2.5: (A) Only the neuron *j=1* fires and its PSP does not overcome the threshold, so the neuron i does not fire. (B) The neurons *j=1* and *j=2* fire at different times. The sum of their PSPs does not overcome the threshold, so the neuron i doesn not fire. (C) Both the neurons *j=1* and *j=2* fire two times. The sum of their PSPs overcome the threshold, so the neuron i fires. [Source: [9]]

pulse indicated by α. Hence, the current is divided into a capacitive and a resistive component:

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt}$$

By multiplying by R, we obtain the membrane time constant τ_m:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t)$$

The neuron generates a spike once the voltage $u(t)$ over the capacitor overcomes the threshold θ. Whenever a spike arrives, the membrane potential is updated by the

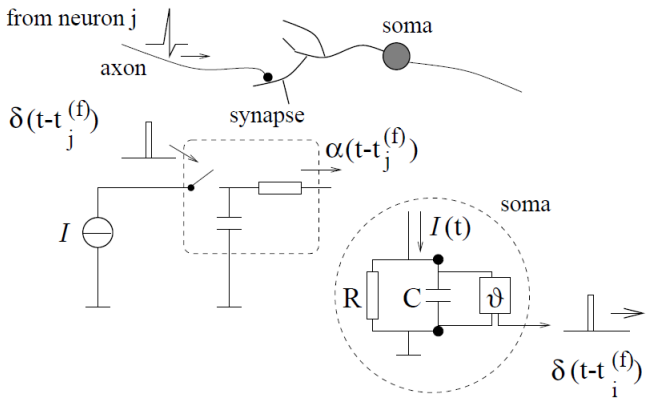


Figure 2.6: Schematic of the IF neuron model. [Source: [9]]

weight of the connection. After the firing, the membrane potential is immediately reset to a value $u_{rest} < \theta$. In a more realistic model, called **Leaky Integrate-and-Fire (LIF)**, the membrane potential decays exponentially.

Izhikevich
This model represents a good trade-off for the computational cost and the similarity to real biological neurons. The model is based on the equations in fig. 2.8 [14].

Comparison
As shown in fig. 2.10, the best trade-off for implementation cost and biological plausibility is the Izhikevich model. LIF model is not present in this analysis, but widely used for its linearity and simplicity. For example, Intel Loihi [16] adopts a variation of this model. The model nearest to the real biological neuron is the Hodgkin-Huxley [9]: it consists of four equations and ten parameters, so it requires a very high number of operations during a simulation. Hence, it is possible to choose a neuron model according to the type of problem and to the number of neurons that we want in our network. In fig. 2.9 we show some neuronal computational properties corresponding to each spiking neuron model: according to the type of input and to the choice of the model parameters, a neuron can figure out a particular behavior (four cases are shown in fig. 2.11).

2.2.3 Information coding strategies

In order to understand how the spikes are encoded in neural informations, many coding strategies have been studied [8]. They are represented in fig. 2.12.

(A) **Time to first spike**: the information is encoded in the latency between the beginning of the stimulus and the first spike. This strategy is very fast and

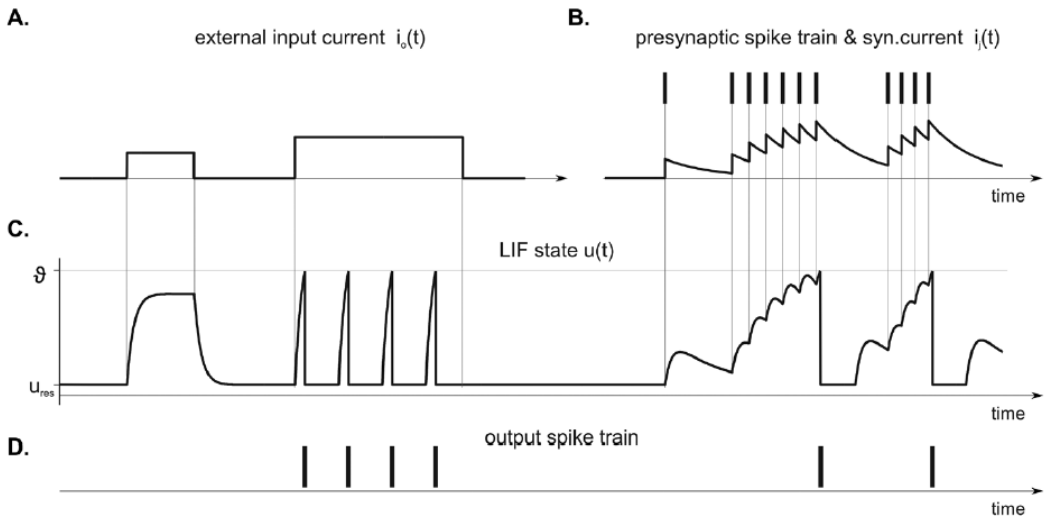


Figure 2.7: (C) shows the behavior of the membrane potential depending on external inputs (A) or presynaptic spike trains (B). (D) shows the firing times. [Source: [8]]

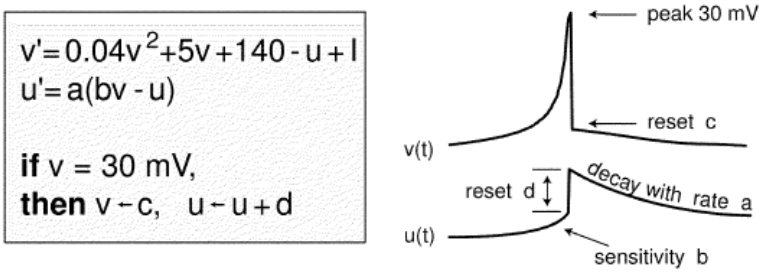


Figure 2.8: Equations of Izhikevich neuron model. u represents a membrane recovery variable, while v the membrane potential. a describes the time scale of the recovery variable. b describes the sensitivity of the recovery variable u to the subthreshold fluctuations of the membrane potential v. c describes the after-spike reset value of the membrane potential v. d describes after-spike reset of the recovery variable u [Source: [14]]

simple.

(B) **Rank-order coding**: the information is encoded considering the order of the spikes. This strategy is simple, assuming that every neuron fires only one time.

(C) **Latency code**: the information is encoded in the latency between spikes. This strategy allows to carry a big amount of information.

(D) **Resonant burst model**: the frequency of a burst determines which neurons have to be activated.

Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-indiced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Figure 2.9: Some features corresponding to each neuron model. [Source: [15]]

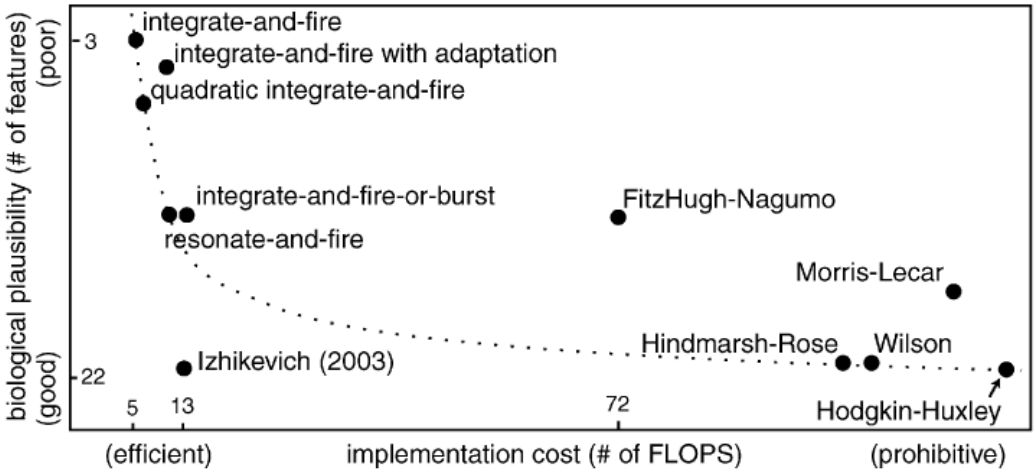


Figure 2.10: Comparison between spiking neuron models. $\#FLOPS$ is the approximated number of floating point operations needed to simulate the model with 1 ms of time span. [Source: [15]]

(E) **Coding by synchrony**: *this model is based on the assumption that neurons*

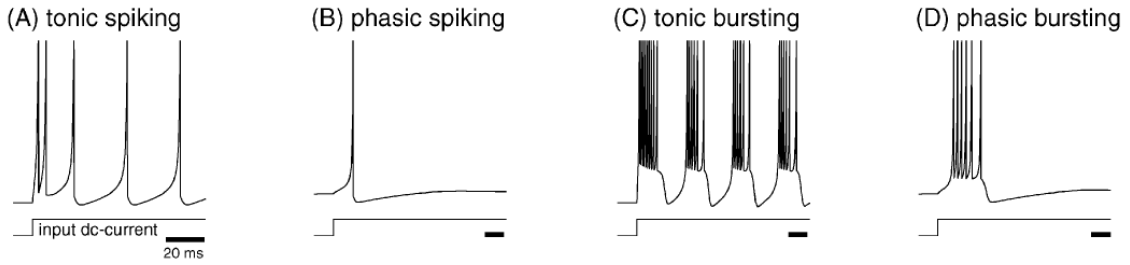


Figure 2.11: Some examples of the spiking behavior of a neuron. (A) While the input is on, the neuron fires a train of spikes. (B) The neuron fires one time when the input goes on. (C) The neuron fires periodic burst of spikes when the input goes on. (D) The neuron fires one burst of spikes when the input goes on. [Source: [15]]

that encode different bits of information on the same object fire synchronously...neuronal synchronization will serve as a mechanism improving, both information transmission through the network, as well as timing precision of spiking events [8].

(F) **Phase coding**: *in this model times of emitted spikes are referred to the reference time point in a periodic signal. In this way neuronal spike trains can encode information in the phase of a pulse with respect to the background oscillations* [8].

2.2.4 SNNs topologies

It is possible to distinguish three different topologies, also valid for DNNs, as showed in fig. 2.13:

1. **Feedforward**: the data go from one layer to another one without feedback connections.

2. **Recurrent**: the connections between layers are not uni-directional in RNNs (Recurrent Neural Networks). These networks, using feedback connections, have higher computational capabilities, *since some intermediate operations generate values that are stored internally in the network and used as inputs to other operations in conjunction with the processing of a later input* [1], and are more difficult to train [8].

3. **Hybrid**: they represent a mix of the previous two cases. An important implementation is the ***Reservoir Computing*** [8]. It takes the advantages of RNNs, but avoids the problems relative to their training phase. We will analyze in depth this important implementation, in particular for SNNs, in the following chapters.

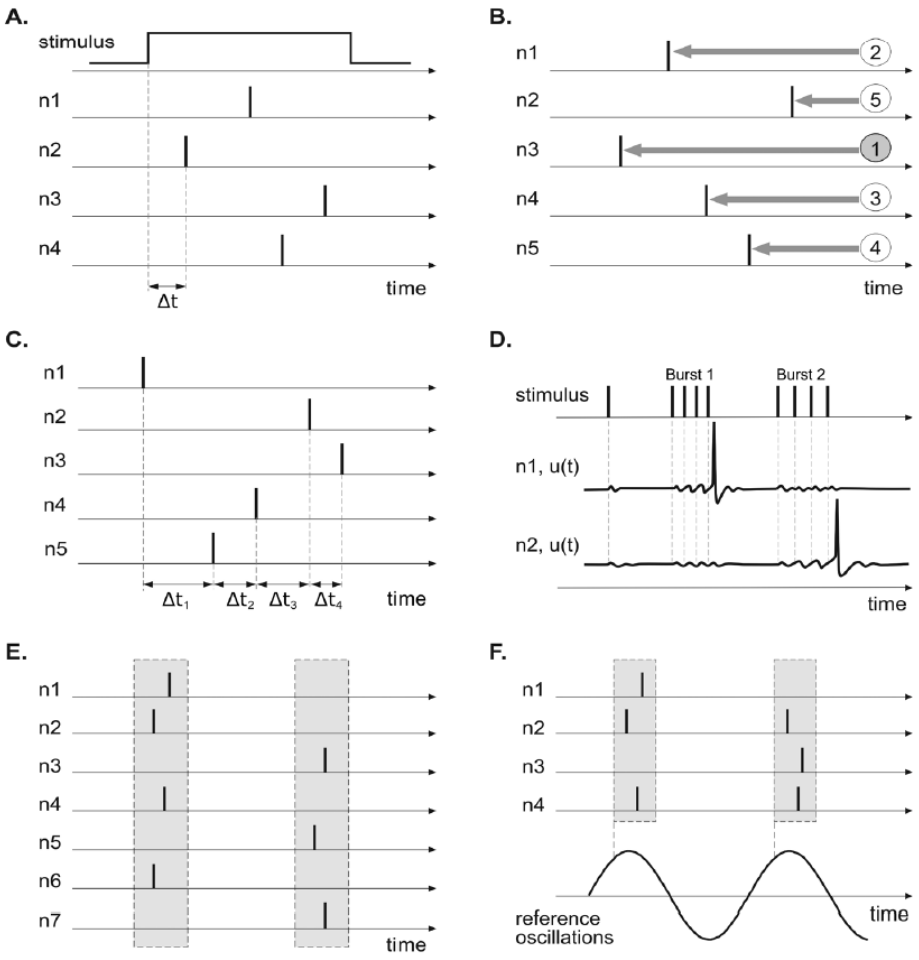


Figure 2.12: Neural coding strategies. (A) Time to first spike. (B) Rank-order coding. (C) Latency code. (D) Resonant burst model. (E) Coding by synchrony. (F) Phase coding. [Source: [8]]

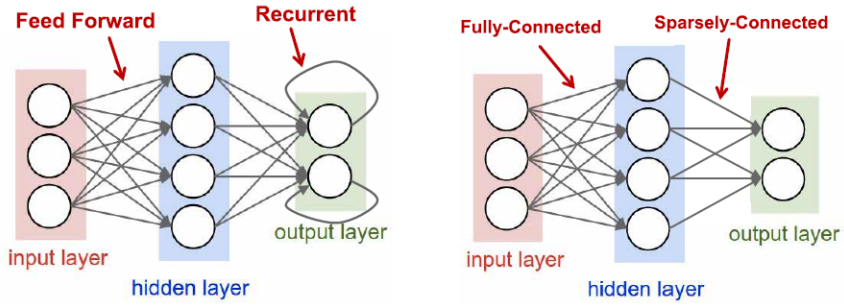


Figure 2.13: Topologies of NNs [Source: [1]]

2.2.5 Spike timing dependent plasticity (STDP)

As for DNNs, for SNNs we can distinguish different types of learning [8]. Since the learning is the modification of the strength of the synapses, we can talk about **synaptic plasticity**.
Synaptic plasticity is the biological process by which specific patterns of synaptic activity result in changes in synaptic strength and is thought to contribute to learning and memory. Both pre-synaptic and post-synaptic mechanisms can contribute to the expression of synaptic plasticity [nature.com].
We briefly focus our background, relative to the learning for SNNs, on the **Spike timing dependent plasticity**. This is a form of **Hebbian learning** [17] based on the timing of the spikes. First of all, we introduce in synthesis the Hebbian learning, focused on the following postulate [17]:
"When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

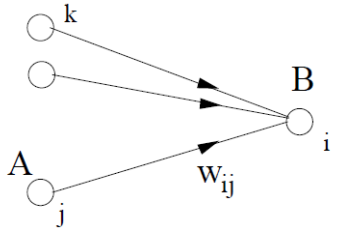


Figure 2.14: The modification of the strength of the synapse depends on the presynaptic and postsynaptic neurons [Source: [9]]

It means that the strength of the synapse w_{ij} depends on the correlation in the firing activity between the neuron j and the neuron i, excluding other neurons k. Hence, if the neurons fire around at the same time, the strenght of the synapse increases. Generally, if a certain synapse is strong, the presynaptic neuron has a strong influence on the output of the postsynaptic neuron. Moreover, in order to understand how a synapse works and to introduce a time dependency, it is important to define two concepts:

- **Long Term Potentiation (LTP)** is a persistent strengthening of the synapses
- **Long Term Depression (LTD)** is a persistent reduction of the synaptic strength

STDP is based on the temporal correlation between the spikes of presynaptic j and postsynaptic i neurons: if the spike of j arrives before than the spike of i, we have a *potentiation*, and the synapse becomes stronger, otherwise we have a *depression*. If

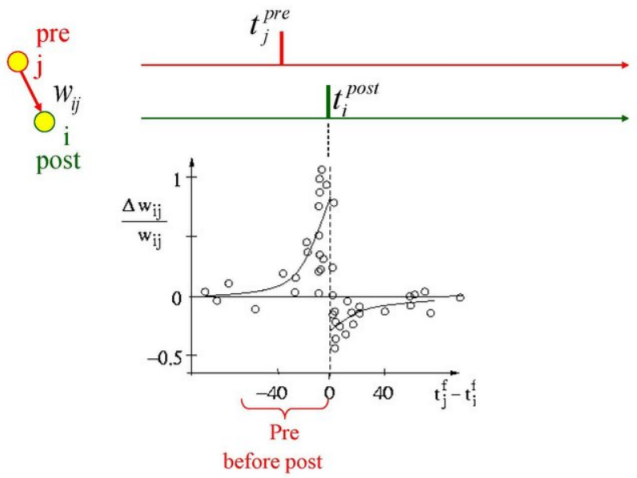


Figure 2.15: STDP function [Source: scholarpedia.com]

the influence of j on i lasts for long time, we have LTP and LTD respectively. To summarize, if the neuron i fires before the spike of j arrives, the strength of the synapse w_{ij} decreases.

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_i^n - t_j^f)$$

$$\begin{cases} W(x) = A_+ exp(-x/\tau_+) \;\; for \;\; x > 0 \\ W(x) = -A_- exp(-x/\tau_-) \;\; for \;\; x < 0 \end{cases}$$

Δw_j is the weight change, t_i^n indicates the firing times of the postsynaptic neurons and t_i^f the firing times of the presynaptic neurons. $W(x)$ is the STDP function (showed in fig. 2.15), based on some variable parameters.

2.2.6 Poisson Spikes Generation and SNNs Applications

SNNs have primarily been used for tasks like real-data classification, biomedical applications, odor recognition, navigation and analysis of an environment, speech and image recognition [7] [8]. One of the most used dataset for image classification is the MNIST database (Modified National Institute of Standards and Technology database) [34] that we will analyze in the next chapter. Recently, the work of [18] proposed to convert every pixel of the images into spike trains (i.e., the sequences of spikes) according to its intensity. The spikes generation can be analyzed under the **Poisson model** [19]. Here we list the main simplified concepts of this theory:

- *Independent spike hypothesis*: we start from the hypothesis that each spike generation is independent of the other spikes.

- Considering the firing rate $r(t)$ and the time interval δt, we have:

$$P\{1 \ spike \ during \ the \ interval \ (t - \delta t, t + \delta t)\} = r(t)\delta t$$

  Consequently, it is possible to demonstrate that the probability of having N spikes in a time T coincides with the Poisson distribution [19].

- The intensity of a pixel is associated to the probability that a spike occurs in a certain time T and the pixels density can be associated to the mean firing rate.

- As [18] suggests, it exists a simple way to create a Poisson spikes generator. At each time step, the program can generate a random variable and compare it with the probability of firing. According to the result of the comparison, we obtain the generation of a spike (if the probability of firing is greater than the random variable), otherwise nothing. Since the firing rate is related to the probability of generating a spike, the pixels with higher values present many spikes, while the pixels with low density present a low number of spikes.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) showed a great improvement and success in many AI applications, e.g., natural language processing, document analysis, face recognition and image classification [2]. These networks achieve high accuracy for object detection, minimizing the classification error. The architecture of a standard CNN is based on four operations, also showed in fig. 2.16:

1. **Convolution**: covering all the pixels of the image with a *filter* or *feature detector*, the *feature map* is obtained. This operation allows to extract some features from the image. Using different filters (characterized by some dimensional parameters), the resulting feature map changes. So using more filters leads to improve the recognition capabilities of the CNN.

2. **Non linearity**: the negative values present in the feature map are replaced with zeros. It is possible to use different types of non linear functions.

3. **Pooling**: the dimensions of the feature map and the parameters of the CNN are reduced, but the most important informations are preserved.

4. **Fully connection**: considering two layers, every neuron belonging to the first layer is connected to every neuron belonging to the second layer.

Many CNNs have been developed during the last years. The fig. 2.17 shows the Top-1 accuracies and other features of many explored CNNs.

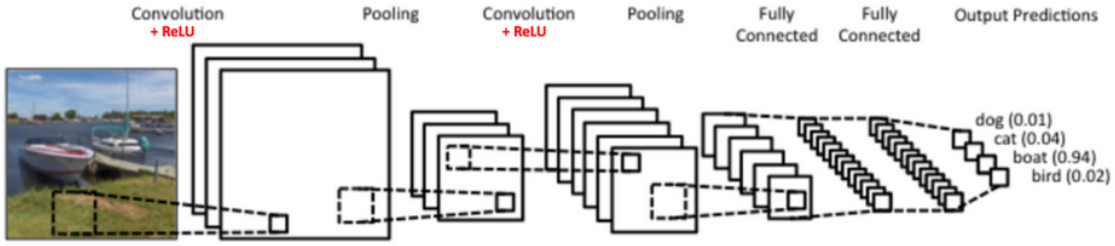


Figure 2.16: Example of a CNN [Source: Clarifai]

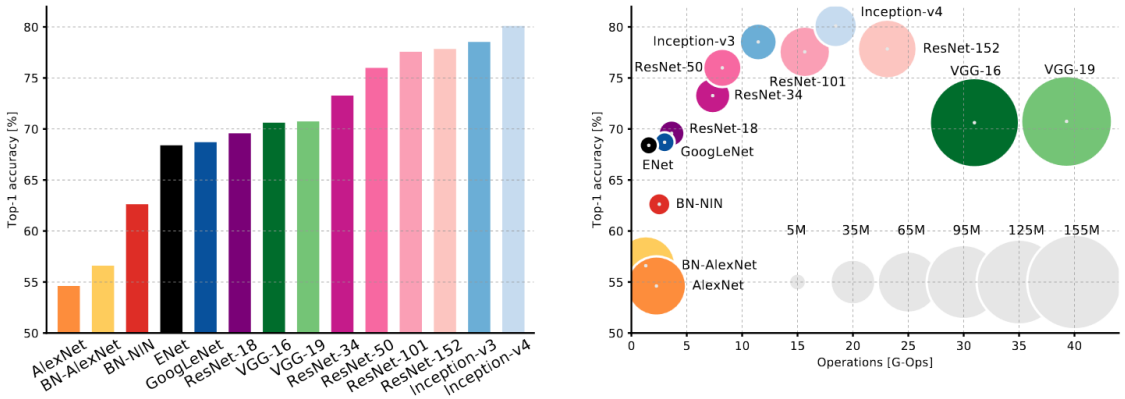


Figure 2.17: (@left) Top-1 accuracy for every CNN (@right) Top-1 accuracy versus the number of operations required for each forward step. The size of the circles is proportional to the parameters of the CNNs. [Source: [3]]

2.4 CapsuleNetworks

CNNs are specialized to identify and recognize the presence of an object as a feature, without taking into account the spatial relationships across multiple features. Recently, Sabour *et al.* [4] proposed **CapsuleNets**, a specialized Neural Network architecture composed of capsules, which is trained based on the *Dynamic Routing algorithm between capsules*. The architecture is shown in fig. 2.18. The key idea behind CapsuleNets is called *inverse graphics*: when the eyes analyze an object, the spatial relationships between its parts are decoded and matched with the representation of the same object in our brain. This concept is called **equivariance**: the properties characterizing an object are contained in its internal representation, that changes according to them. Hence, CapsuleNets are able to recognize the position of an object relative to another one, unlike the CNNs. Similarly, in CapsuleNets the feature representations are stored inside the capsules in a vector form, in contrast to the scalar form used by the neurons in traditional Neural Networks [5]. Sabour *et*

al [4] defined a capsule as *a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part*. The capsules operate in a way similar to the one performed by the artificial neurons: they sum the weighted inputs (previously the input vectors are multiplied by the weight matrices) and then they apply the non linearity (using a *squashing function*). Moreover, thanks to the use of vectors instead of scalars, CapsuleNets can obtain a good accuracy using less training data with respect to the CNNs.

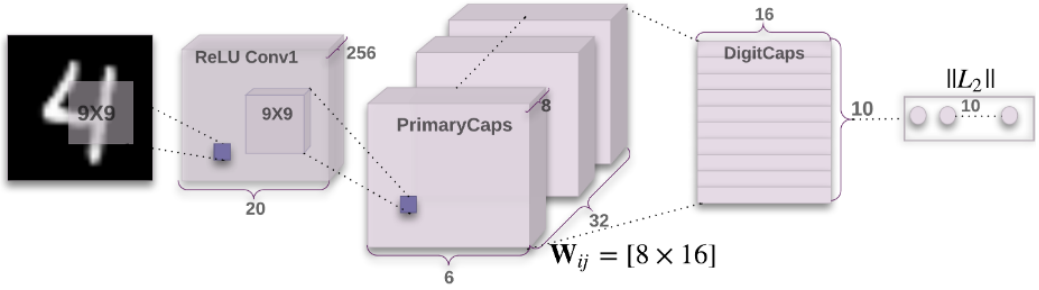


Figure 2.18: Architecure of CapsuleNet. [Source: [4]]

The architecture is composed of three layers for the encoding and three fully connected layers for the decoding:

1. **Convolutional layer**: input 28x28, output 20x20x256
2. **PrimaryCaps layer**: output 6x6x8x32
3. **DigitCaps layer**: output 16x10
4. **First FC layer**: output 512
5. **Second FC layer**: output 1024
6. **Third FC layer**: output 784

Recent researches about CapsuleNet architecture and training algorithms [4] [6] have shown competitive results, in terms of accuracy, for image classification task, compared to other state-of-the-art classifiers.

2.5 Adversarial Attacks

2.5.1 Image classification

Generally a dataset is composed by three different sets:

1. **Training set** to adjust the weights
2. **Validation set** to minimize overfitting
3. **Test set**, composed of images not previously used, to test the accuracy of the model

The task of the **image classification** is to identify an image given as input. As shown in fig. 2.19, the model answers with some probabilities associated to each class (six in this case, the *top-5 accuracy* is the most used standard, presenting the top five probabilities). In our example, the image is labeled as a dog. If we use a supervised learning, the classification is correct when the output class corresponding to the maximum probability corresponds to the label. According to the correctly predicted images, a certain **accuracy** computed in percentage is associated to the neural classifier model. For example, if the test set is composed of 10000 images and our model classifies correctly 9000 images, we obtain an accuracy of 90%. What is the difference between image classification/recognition and **object detection**? In object detection, an image localization is applied to all the objects present in the image and every object is boxed, as shown in fig. 2.20. Why is image recognition so important today? We use image recognition in many applications, for example face recognition and security systems, medical applications, robots, object tracking or autonomous driving.

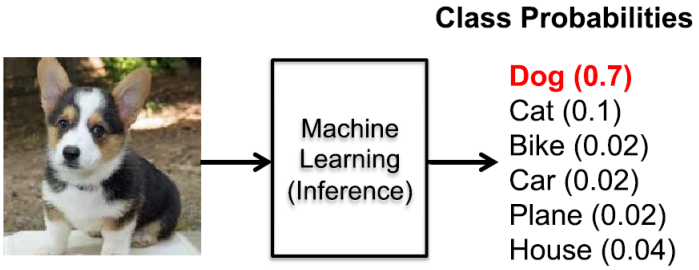


Figure 2.19: Example of image classification [Source: [1]]

2.5.2 Adversarial examples - basic knowledge

*Some machine learning models are vulnerable to adversarial perturbations. An **adversarial example** is a sample of input data which has been modified very slightly in a way that is intended to cause a machine learning classifier to misclassify it. In many cases, these modifications can be so imperceptible that a human observer does not even notice the modification at all, yet the classifier still makes a mistake* [22].

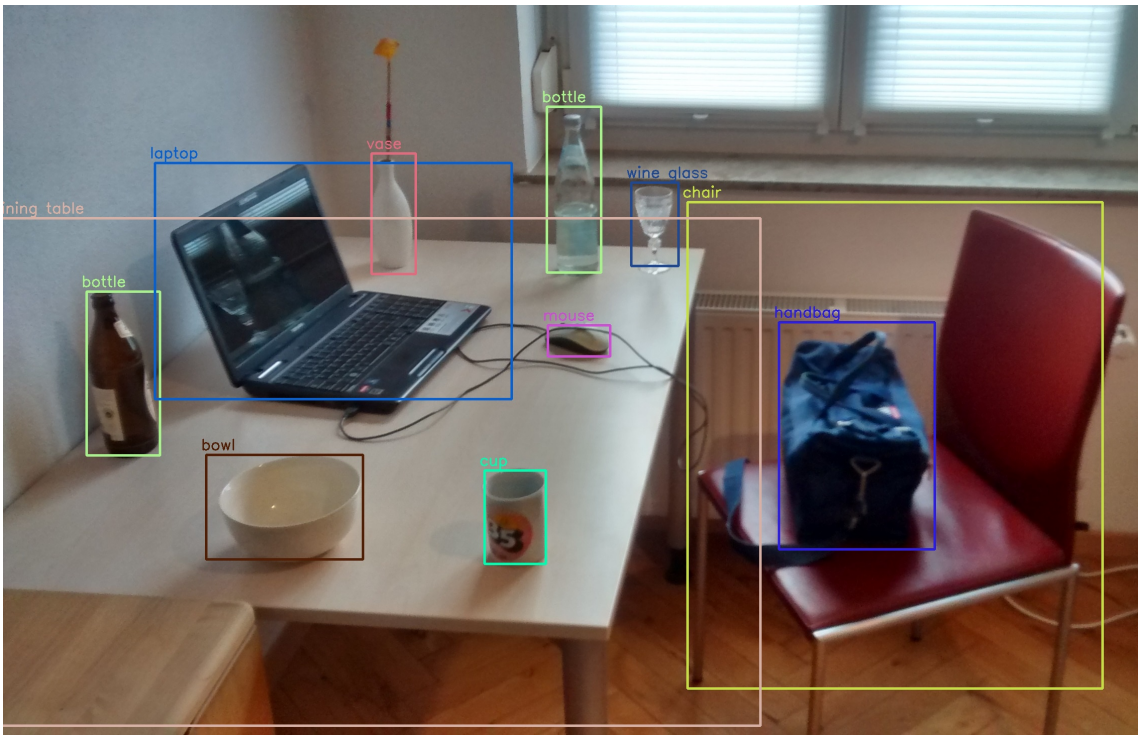


Figure 2.20: Example of object detection [Source: Wikipedia]

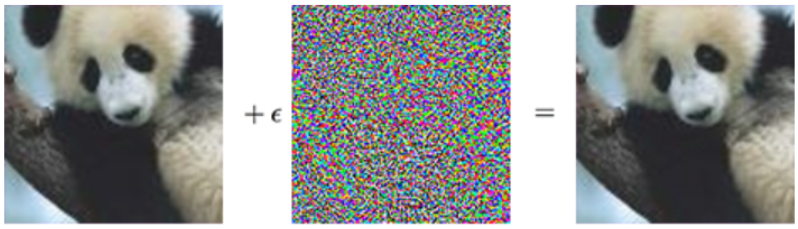


Figure 2.21: Example of an adversarial attack. A panda is misclassified into a gibbon. [Source: openai.com]

Adversarial examples are small perturbations added to the inputs, which are generated for the purpose to mislead the network. Szegedy *et al.* [30] were the first to discover that several machine learning models are vulnerable to adversarial examples. Goodfellow *et al.* [21] explained the problem observing that *machine learning models misclassify examples that are only slightly different from correctly classified examples drawn from the data distribution.* A minimal and imperceptible modification of the input data can cause a classifier misprediction, which can potentially produce

a wrong output with high probability, as we can observe in fig. 2.21. Since these examples can fool the network and point out its security vulnerability, they can be dangerous in safety-critical applications, like automotive, medical, privacy and banking applications, Voice Controllable Systems (VCS) [35] [36] and traffic signs recognition [22] [31]. For instance, in the image recognition field, having a wide variety of possible real world input images [22], with high-complex pixel intensity patterns, the classifier cannot recognize if the source of the misclassification is the attacker or other factors [20]. Many works [13] [20] [21] [22] [23] [27] [28] [29] [32] have analyzed the impact of adversarial examples in Neural Networks and studied different methodologies to improve the defense solutions.

2.5.3 Adversarial types

Adversarial attacks can be categorized according to different properties, e.g., the choice of the class, the kind of the perturbation and the knowledge of the network under attack. Given an input image x, the goal of an adversarial attack $x^* = x + \delta$ is to apply a small perturbation δ such that the predicted class $C(x)$ is different from the target one $C(x^*)$, i.e., the class in which the attacker wants to classify the example. Inputs can also be misclassified without specifying the target class: this is the case of **untargeted** attacks, where the target class is not defined a-priori by the intruder. **Targeted** attacks can be more difficult to apply than the untargeted ones, but they are more efficient. *Individual* attacks create perturbations of different magnitude for each different input, while *universal* attacks apply the same perturbation to all the inputs of the dataset [31]. Attacks are applied under **black box** assumption when the attacker does not know the architecture, the training data and the parameters of the network [25]. When the attacker knows the architecture, the training data and the parameters of the network, the attack is under **white box** assumption.
Another classification for adversarial attacks is showed in fig. 2.23. Another one is the following [33]:

- **Gradient-based**: they depend on the gradient of the loss function with respect to the input.
- **Score-based**: they depend on the output probabilities of the model.
- **Transfer-based**: they depend on the training data that are used to train a substitute model used to craft the adversarial examples.
- **Decision-based**: they depend on the final decision of the model, i.e., the top-1 class label.

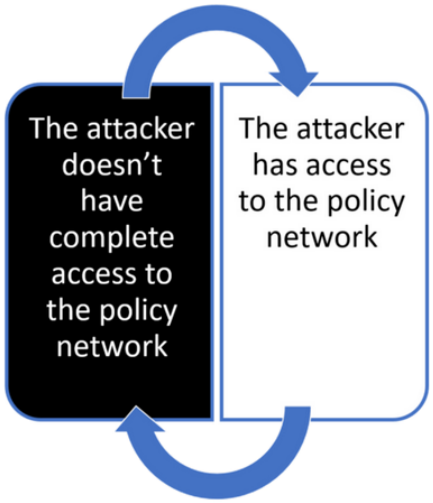


Figure 2.22: Difference between black and white box assumptions. [Source: towardsdatascience.com]

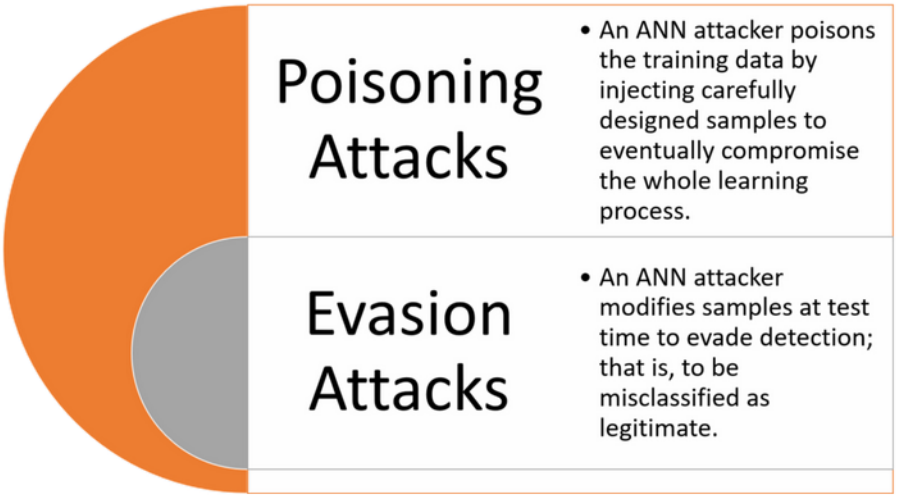


Figure 2.23: Another classification for adversarial attacks. [Source: towardsdatascience.com]

Chapter 3

SNN under Attack: are Spiking Deep Belief Networks vulnerable to Adversarial Examples?

In this chapter, we aim at generating, for the first time, imperceptible and robust adversarial examples for SNNs. For the evaluation, as a case study we apply these attacks to a Spiking Deep Belief Network (SDBN) and a DNN having the same number of layers and neurons, to obtain a fair comparison. As per our knowledge, this kind of attack was previously applied **only** on a DNN model [26]. This method is efficient for DNNs because it is able to generate adversarial examples which are imperceptible to the human eye, as compared to the original image. Moreover, in the physical world, the attack efficacy can significantly decrease if the pre-processing transformations such as compression, resizing, noise filtering are applied to the input images [26] [39]. In this chapter, we investigate the vulnerability of SDBNs to random noise and adversarial attacks, aiming at identifying the similarities and the differences with respect to DNNs. Our experiments show that, when applying a random noise to a given SDBN, its classification accuracy decreases, by increasing the noise magnitude. Moreover, applying our attack to SDBNs, we observe that, in contrast as the case of DNNs, the output probabilities follow a different behavior: while the adversarial image remains imperceptible, the misclassification is not always guaranteed. Note, our attack can be applied not only to SNNs and DNNs, but to every type of NNs, under black-box assumptions.

Our Novel Contributions:

1. We analyze how the SDBN accuracy varies when a random noise is added to the input images (Section 3.2).

2. We evaluate the improved generalization capabilities of the SDBN when adding a random noise to the training images (Section 3.3).

3. We develop a novel methodology to automatically create imperceptible adversarial examples for every type of NNs. (Section 3.4).

4. We apply our methodology to a DNN and an SDBN *(it is the first attack of this type applied to SDBNs)*, and evaluate the imperceptibility and the robustness of the adversarial examples (Section 3.5).

Before proceeding to the technical sections, in the Section 3.1 we briefly review some works related to our paper, focusing on SDBNs and adversarial attacks for DNNs.

3.1 Related Works

3.1.1 Spiking Deep Belief Networks

Deep Belief Networks (DBNs) [45] are multi-layer networks that are widely used for classification problems and implemented in many areas such as visual processing, audio processing, images and text recognition with optimal results [45]. DBNs are implemented by stacking pre-trained Restricted Boltzmann Machines (RBMs), energy-based models consisting in two layers of neurons, one hidden and one visible, symmetrically and fully connected, i.e., without connections between the neurons inside the same layer (this is the main difference with respect to the standard Boltzmann machines). RBMs are typically trained with unsupervised learning, to extract the information saved in the hidden units, and then a supervised training is performed to train a classifier based on these features [46]. Spiking DBNs (SDBNs) improve the energy efficiency and computation speed, as compared to DBNs. Such behavior has already been observed by O'Connor [38]. O'Connor *et al.* [38] proposed a DBN model composed by 4 RBMs of 784-500-500-10 neurons, respectively. It has been trained offline and transformed in an event-based domain to increase the processing efficiency and computational power. The RBMs are trained with the Persistent Contrastive Divergence (CD) algorithm, an unsupervised learning rule using Gibbs sampling, a Markov chain Monte Carlo algorithm, with optimizations for fast weights, selectivity and sparsity [42] [43] [44]. Once every RBM is trained, the information is stored in the hidden units to use it as input for the visible units of the following layer. Afterwards, a supervised learning algorithm [41], based on the features coming from the unsupervised training, is performed. The RBMs of this model use the *Siegert* function [47] in their neurons. It allows to have a good approximation of firing rate of Leaky Integrate and Fire (LIF) neurons [9], used for CD training. So in a SDBN the neurons generate Poisson spike trains according to the Siegert formula: this represents a great advantage in terms of power consumption and speed, as compared to classical DBNs, which are based on a discrete-time model [38].

3.1.2 Imperceptible and Robust Adversarial Attacks

Luo *et al.* [26] propose a new method to generate attacks maximizing their noise tolerance and taking into account the human perceptual system in their distance metric. This methodology has strongly inspired our algorithm. Since the human eyes are more sensitive to the modifications of the pixels in low variance areas, to maintain as much as possible the imperceptibility it is preferable to modify the pixels in high variance areas. From the other side, a robust attack aims to increase *its ability to stay misclassified to the target class after the transformations due to the physical world*. For example, considering a crafted sample, after an image compression or a resizing, its output probabilities can change according to the types of applied transformations. Therefore, the attack can be ineffective if it is not robust enough to those variations. Motivated by these considerations, we propose an algorithm to automatically generate imperceptible and robust adversarial examples.

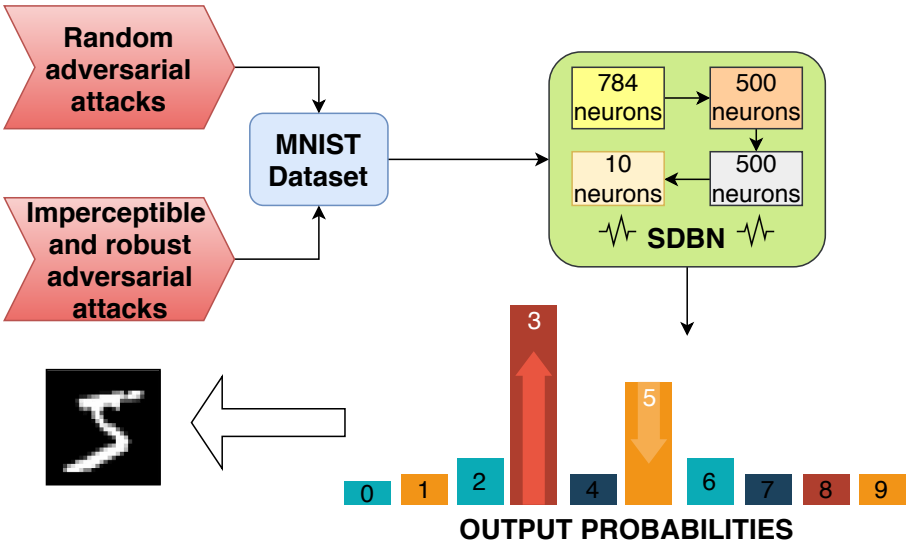


Figure 3.1: Overview of our approach

3.2 Analysis: applying random noise to SDBNs

3.2.1 Experiment Setup

We take as a case-study example an SDBN [Neil, 2013] composed by four fully-connected layers of 784-500-500-10 neurons, respectively. We implement this SDBN in Matlab, for analyzing the MNIST database, a collection of 28·28 gray scale images of handwritten digits, divided into 60.000 training images and 10.000 test images.

ACC	TRAIN	TEST	TR+TST	TRAIN	TEST	TR+TST
δ	NORMALLY			UNIFORMLY		
0.02	**96.65**	94.73	96.54	**96.8**	96.02	96.81
0.05	95.19	94.42	94.99	96.7	95.64	96.72
0.08	92.99	82.73	73.64	95.89	94.64	95.56
0.1	76.01	77.07	10.39	94.34	93.36	92.8
0.15	24.61	**48.23**	10.32	47.03	**82.76**	10.51
0.2	10.26	33.34	10.05	14.64	60.79	10.16
0.3	10.31	21.52	9.88	9.59	34.9	10.16
0.4	10.27	17.05	10.34	9.98	23.16	10.03

Table 3.1: Evaluation of SDBN accuracy applying two different types of random noise with different values of noise magnitude.

Each pixel is encoded as a value between 0 and 255, according to its intensity. To maximize the spike firing, the input data are scaled to the range [0-0.2], before converting them into spikes. In our simulations, the pixel intensities are represented as the probability that a spike occurs.

3.2.2 Understanding the Impact of Random Noise Addition to Inputs on the Accuracy of an SDBN

We test the accuracy of our SDBN for different noise magnitudes, applied to three different combinations of images:

- to all the training images.
- to all the test images.
- to both the training and test images.

In order to test the vulnerability of our SDBN, we apply two different types of noises: normally distributed and uniformly distributed random noise.

The results of the experiments are shown in Table 3.1 and Figure 3.2. The starting accuracy, obtained without applying noise, is 96.2%. When the noise is applied to the test images, the accuracy of the SDBN decreases accordingly with the increasing of the noise magnitude, more evidently in the case of normally distributed random noise. This behavior is due to the fact that the standard normal distribution contains a wider range of values, compared to the uniform one. For both noise distributions, the accuracy decreases more when the noise magnitude applied lays around 0.15 (see the red-colored values in Table 3.1).

When the noise is applied to the training images, the accuracy of the SDBN does not decrease as much as in the previous case, as long as the noise magnitude is lower than 0.1. On the contrary, for $\delta = 0.02$, the accuracy increases (see the green-colored values in Table 3.1). with respect to the baseline, without noise. Indeed, adding noise in training samples improves the generalization capabilities of the neural network. Hence, its capability to correctly classify new unseen samples also increases. This observation, already analyzed in several other scenarios for Deep Neural Networks with back-propagation training [40], is also valid for our SDBN model. However, if the noise is equal to or greater than 0.1, the accuracy drops significantly: this behavior means that the SDBN is learning noise instead of useful information, thus it is not able to classify correctly.

When the noise is applied to both the training and test images, we can notice that the behavior observed for the case of noise applied to the training images only is accentuated: for low values of noise magnitude (mostly in the uniform noise case) the accuracy is similar or higher than the baseline; for noise magnitudes greater than 0.1 (more precisely, 0.08 for the case of normal noise applied), the accuracy decreases more sharply than in the case of noise applied to the training images only.

3.2.3 Applying Noise to a Restricted Window of Pixels

Further analyses have been performed: we add a normally distributed random noise to a restricted window of pixels of the test images. Considering a rectangle of 4x5 pixels, we analyze two scenarios:

- The noise is applied to 20 pixels at the top-left corner of the image. The variation of the accuracy is represented by the blue-colored line of Figure 3.3. As expected, the accuracy remains almost constant, because the noise affects irrelevant pixels. The resulting image, when the noise is equal to 0.3, is shown in Figure 3.4b.

- The noise is applied to 20 pixels in the middle of the image, with coordinates $(x, y) = ([14\ 17], [10\ 14])$. The accuracy descreases more significantly (orange-colored line of Figure 3.3), as compaerd to the previous case, because some white pixels representing the handwritten digits (and therefore important for the classification) are affected by the noise. The resulting image, when the noise is equal to 0.3, is shown in Figure 3.4c.

3.2.4 Key Observations from our Analyses

From the performed analyses, we derive the following key observations:

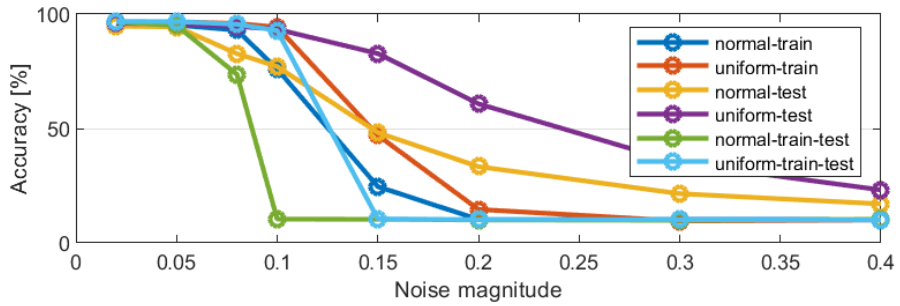


Figure 3.2: Normal and uniform random noise applied to all the pixels of the MNIST dataset.

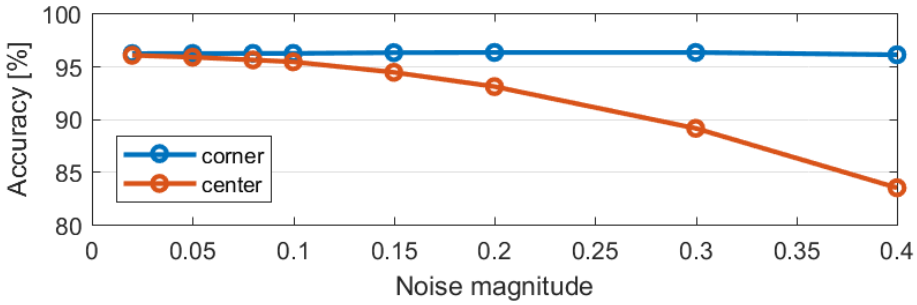


Figure 3.3: Normal random noise applied to some pixels of the MNIST dataset test images.

- The normal noise is more powerful than the uniform counterpart, since the accuracy decreases more sharply.
- For a low noise magnitude applied to the training images, we notice a small accuracy improvement, due to the improved generalization capability of SDBNs.
- When applying the noise to a restricted window of pixels, the perturbation is more effective if the window is in the center of the image, as compared to the corner, because the noise is applied to the pixels which are relevant for the classification.

3.3 Our novel methodology to generate imperceptible and robust adversarial attacks

The scope of a good attack is to generate adversarial images, which are difficult to be detected by human eyes and resistant to physical transformations. Therefore, for

(a) (b) (c)

Figure 3.4: Comparison between images with normally distributed random noise (with magnitude 0.3) applied to the corner and to the left center of the image. (a) Without noise. (b) Noise applied to the top-left corner. (c) Noise applied to the center of the image.

better understanding this challenge, we first analyze two concepts: imperceptibility and robustness.

3.3.1 Imperceptibility of adversarial examples

Creating an imperceptible example means to add perturbations to some pixels, while being aware to make sure that humans do not notice them. We consider an area A=N·N of pixels x, and we compute the standard deviation (SD) of a pixel $x_{i,j}$ as in Equation (5.1):

$$SD(x_{i,j}) = \sqrt{\frac{\sum_{k=1}^{N}\sum_{l=1}^{N}(x_{k,l}-\mu)^2 - (x_{i,j}-\mu)^2}{N \cdot N}}, \tag{3.1}$$

where μ is the average value of pixels belonging to the N·N area. If a pixel has a high standard deviation, it means that a perturbation added to this pixel is more likely to be hardly detected by the human eye, compared to a pixel with low standard deviation. The sum of all the perturbations δ added to the pixels of the area A allows to compute the distance ($D(X^*, X)$) between the adversarial example X^* and the original one X. Its formula is shown in Equation (3.2).

$$D(X^*, X) = \sum_{i=1}^{N}\sum_{j=1}^{N}\frac{\delta_{i,j}}{SD(x_{i,j})} \tag{3.2}$$

Such value can be used to monitor the imperceptibility: indeed, the distance $D(X^*, X)$ indicates how much perturbation is added to the pixels in the area A. Hence, the maximum perturbation tolerated by the human eye can be associated to a certain value of the distance, D_{MAX}.

3.3.2 Robustness of adversarial examples

Another important concept to analyze is the robustness. Many adversarial attack methods used to maximize the probability of target class to ease the classifier misclassification of the image. The main problem of this methods is that they do not take in account the relative difference between the class probabilities, i.e., the gap, defined in Equation (5.3).

$$Gap(X^*) = P(targetclass) - max\{P(otherclasses)\} \tag{3.3}$$

Therefore, after an image transformation, a minimal modification of the probabilities can make the attack ineffective. In order to improve the robustness, it is desirable to increase the difference between the probability of the target class and the highest probability of the other classes. In other words, to maximize the gap function.

3.3.3 How to automatically generate attacks

Considering these important parameters, we designed a novel greedy algorithm that automatically generates adversarial examples imperceptible and robust. This algorithm is based on the black-box assumption: the attacks are performed on some pixels of the image, thereby without needing to know the insights of the network. Given the maximum allowed distance D_{MAX} such that human eyes cannot detect perturbations, the problem can be expressed as in Equation (3.4).

$$\arg\max_{X^*} Gap(X^*) \;\mid\; D(X^*, X) \leq D_{MAX} \tag{3.4}$$

In summary, *the purpose of our iterative algorithm is to perturb a set of pixels, to maximize the gap function, thus making the attack robust, while keeping the distance between the samples below the desired threshold, in order to remain imperceptible.* Our iterative algorithm perturbs only a window of pixels of the total image. We choose a certain value N, which corresponds to an area of N·N pixels, performing the attack on a subset M of pixels. Our proposed methodology to automatically generate adversarial examples is shown in Algorithm 1. After having computed the standard deviation for the selected N·N pixels, we compute the gap function, i.e., the difference between the probability of the target class and the highest probability between the other classes. Then, the algorithm decides whether to apply a positive

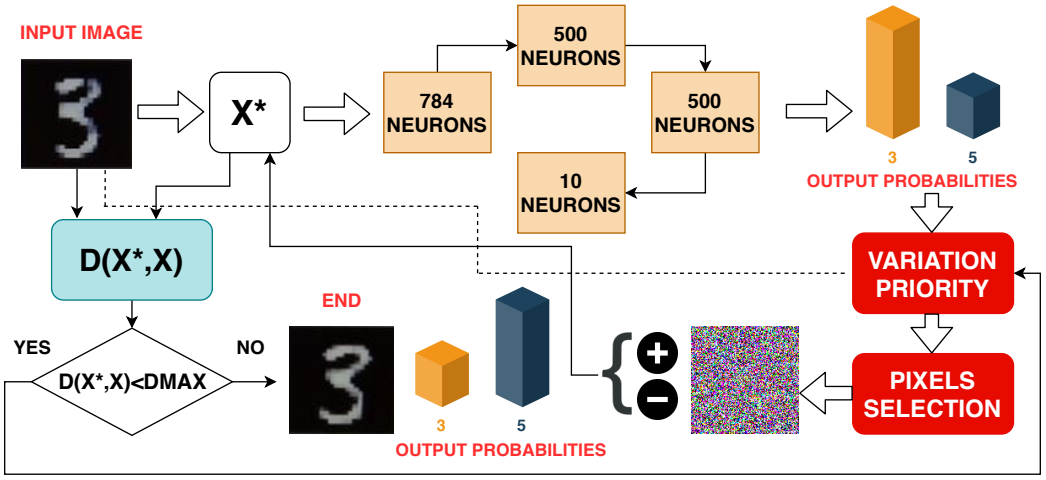


Figure 3.5: Our methodology for generating adversarial examples.

or negative noise to the pixels. Therefore, we compute two parameters for each pixel, $Gap^{+}(X^{*})$ and $Gap^{-}(X^{*})$. $Gap^{+}(X^{*})$ is the value of the gap function computed by adding a perturbation unit to the single pixel, while $Gap^{-}(X^{*})$ is the counterpart, computed subtracting a perturbation unit. According to the difference between these values and the gap function, and considering also the standard deviation, we compute the variation priority, a function that indicates the effectiveness of the pixel perturbation. For example, if $Gap^{-}(X^{*})$ is greater than $Gap^{+}(X^{*})$, it means that, for that pixel, subtracting the noise will be more effective than adding it to the pixel, since the difference between $P(target\ class)$ and $max[P(other\ classes)]$ will increase more. Once computed the vector *VariationPriority*, its values are ordered and the highest M values are perturbed. Note: according to the previous considerations, the noise is added to or subtracted from the selected M pixels, depending on the highest value between $Gap^{+}(X^{*})$ and $Gap^{-}(X^{*})$. The algorithm starts the next iteration by replacing the original input image with the created adversarial one. The iterations terminate when the distance between original and adversarial examples overcomes the maximum perceptual distance. Figure 3.5 summarizes our algorithm for generating adversarial examples.

3.4 Evaluating our attack on SDBNs and DNNs

3.4.1 Setup

By using our methodology, described in Section 4.3, we attack two different networks: the same SDBN as the one analyzed in Section 3 and a DNN, both implemented in Matlab. Note, to achieve a fair comparison, we design the DNN for our experiments

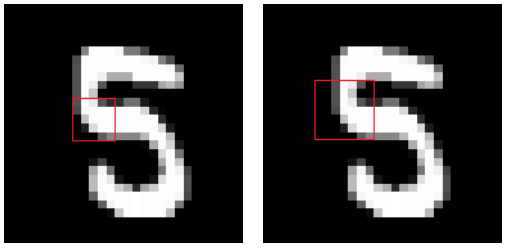
Algorithm 1 Algorithm 1: Our Methodology

Given: original sample X, maximum human perceptual distance D_{max}, noise magnitude δ, area of A pixels, target class, M
while $D(X^*, X) < D_{MAX}$ **do**
-Compute *Standard Deviation SD* for every pixel of A
-Compute $Gap(X^*)$, $Gap^-(X^*)$, $Gap^+(X^*)$
if $Gap(X^*)^- > Gap(X^*)^+$ **then**
VariationPriority($x_{i,j}$)=$[Gap^-(X^*) - Gap(X^*)] * SD(x_{i,j})$
else
VariationPriority($x_{i,j}$)=$[Gap^+(X^*) - Gap(X^*)] * SD(x_{i,j})$
end if
-Sort in descending order VariationPriority
-Select M pixels with highest VariationPriority
if $Gap(X^*)^- > Gap(X^*)^+$ **then**
Subtract noise with magnitude δ from the pixel
else
Add noise with magnitude δ to the pixel
end if
-Compute $D(X^*, X)$
-Update the original example with the adversarial one
end while

having the same architecture as the SDBN, i.e., composed by four fully-connected layers of 784-500-500-10 neurons, respectively. The DNN is trained with scaled conjugate gradient backpropagation algorithm and its classification accuracy of MNIST dataset is 97.13%. In order to test our methodology, we select a test sample, labeled as five (see Figure 3.6). It is classified correctly by both networks, but with different output probabilities. We use a value of δ equal to the 10% of the pixel scale range and a D_{MAX} equal to 22 to compare the attacks. We distinguish two cases, having different search window sizes:

1. Figure 3.6a: N=5 and M=10. Motivated by the experiments performed in Section 3.2, we define the search window in a central area of the image, as shown by the red square of 3.6a.

2. Figure 3.6b: N=7 and M=10. It can be interesting to observe the difference with respect to the case I: in this situation we perturb the same amount M of pixels, selected from a search window which contains 24 more pixels.



(a) (b)

Figure 3.6: Selected area of pixels to attack

CASE	ITER	P MAX CLASS	P TARGET CLASS	DISTANCE
I	0	**98.79**	**0.89**	0
I	14	**44.16**	**55.74**	20.18
I	15	**36.25**	**63.67**	21.77
II	0	**98.79**	**0.89**	0
II	10	**57.53**	**42.01**	16.29
II	11	**49.45**	**50.32**	21.19

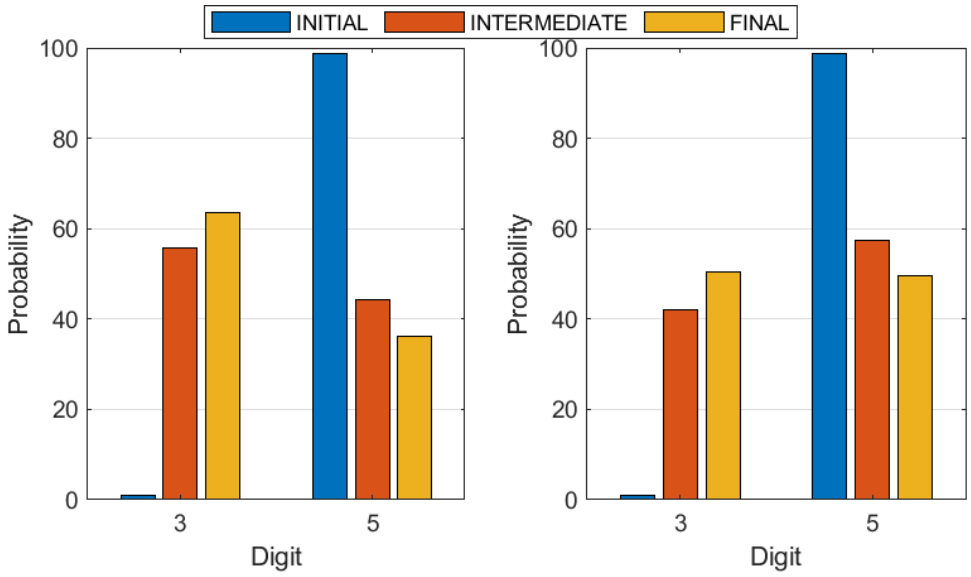
Table 3.2: Results of our simulations for the DNN.

(Case I) After 14 iterations, the probability of the target class has overcome the one of the initial class. The Figure 3.8a shows the sample at this stage (denoted as *intermediate* in Figure 3.7a). At the following iteration, the gap between the two classes increases, thus increasing the robustness of the attack, but also increasing the distance. The sample at this point (denoted as *final* in Figure 3.7a) corresponds to the output of the attack, since at the iteration 16 the distance falls above the threshold.

(Case II) After 11 iterations (denoted as *final* in Figure 3.7b), the sample (in Figure 3.8d) is classified as a three. Since at the iteration 12 the distance is already higher than D_{MAX}, we show in Figure 3.8c the sample at the 10^{th}, whose output probabilities are denoted as *intermediate* in Figure 3.7b.

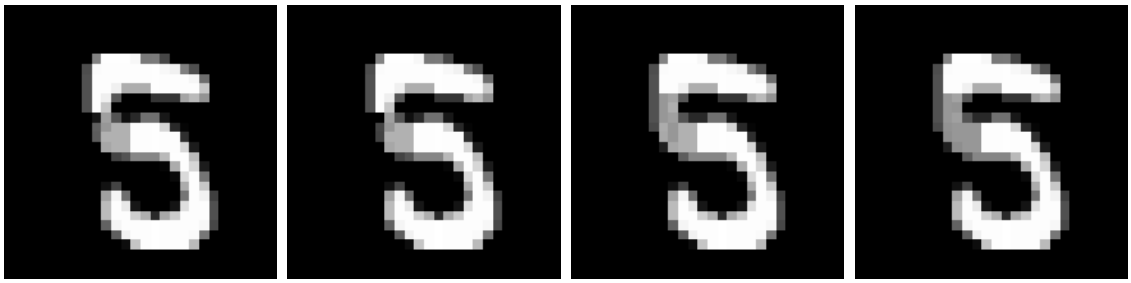
3.4.2 DNN Under Attack

The baseline DNN classifies our test sample as a five with its associated probability equal to 98.79%, as shown in the blue-colored bars of Figure 3.7. The selected target class is three for both the cases. The classification results of their respective adversarial images are showed in Figure 3.7 for both the cases. From the results in Table 3.2 we can observe that, having a small search window leads to obtaining a more robust attack, as compared to larger search windows.



(a) (b)

Figure 3.7: Output probabilities, in percentage format, of the DNN for the crafted sample. (a) Attack using the search window of case I. (b) Attack using the search window of case II.



(a) (b) (c) (d)

Figure 3.8: Adversarial samples applied to the DNN. (a) 14^{th} iteration of case I. (b) 15^{th} iteration of case I. (c) 10^{th} iteration of case II. (d) 11^{th} iteration of case II.

3.4.3 SDBN Under Attack

The baseline SDBN, without attack, classifies our test sample as a five with a probability equal to 82.69%. The complete set of output probabilities is shown in Figure 3.9. We select the three as the target class. In contrast to the attack applied to the DNN, we observe, for the case I, that:

- The set of the SDBN output probabilities does not change monotonically when increasing the iterations of our algorithm.

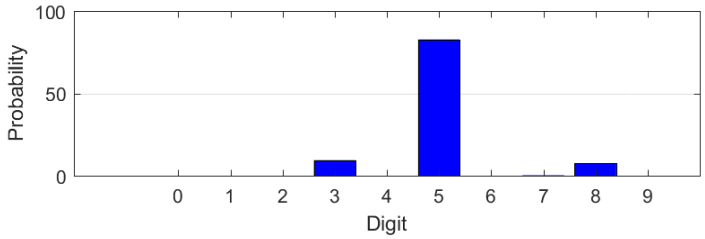


Figure 3.9: Output probabilities of the SDBN for the original sample

- At the 20^{th} iteration, the SDBN still classifies the target class with a probability of 31.08%, while $D(X^*, X) = 7.79$.
- At the other iterations, before and after iteration 20, the output probability of classifying the image as a five still dominates.

Meanwhile, for the case II, we observe that:

- At the 9^{th} iteration, the SDBN misclassifies the image: the probability of classifying a three is 50.60%, with a distance $D(X^*, X) = 10.91$. As a side note, the probability of classifying a eight is 49.40%.
- At the other iterations, before and after the iteration 7, the output probability of classifying the image is higher than 50%.

3.4.4 Comparison

We can observe how DNNs are vulnerable to the attacks generated by our algorithm, while the SDBN shows a very particular behavior. They do not follow the expected trend, but may sporadically lead to a misclassification if also other conditions (that we did not consider in our model analysis) are satisfied. Each pixel of the image is converted as a spike train, thus a slight modification of the pixel intensity can have unexpected consequences. The SNN sensitivity of the targeted attack is clearly different from the respective DNN sensitivity. Such difference of robustness should be studied more carefully in future researches.

3.5 Conclusions

In this paper, we have not answered the fundamental questions that we raised. The SNN vulnerability/robustness to adversarial attacks still needs to be investigated further. However, we opened new research questions that need to be addressed in the future work. What is hidden inside the SNNs that make them more robust

to targeted attacks, as compared to DNNs? Are their computational similarities to the human brain the mean towards robust machine learning? Thus, extensive in-depth studies of SNNs may bear the potential to adopt ML-based solutions even in safety-critical applications.

Chapter 4

Vulnerability of LSMs to imperceptible and robust adversarial examples

Spiking Neural Networks (SNNs) represent the third generation of Neural Networks [10]. These networks perform a great step forward in the direction of studying an architecture increasingly similar to the human brain. The spiking neurons communicate by generating and propagating sequences of spikes, called spike trains, whose difference in time allows to encode the information in an high energy efficient way [12]. Moreover, networks of spiking neurons, compared to the networks of the first two generations of Neural Networks, require fewer neurons to make the same computations: SNNs show a great potential from the computational point of view [11] [12]. In many applications in which SNNs are used, e.g., tracking systems, decision making and action selection [8], it is fundamental to take into account not only the input data, but also the temporal information. In order to process this kind of information, it has been demonstrated that recurrent connections in Neural Networks enable to deal with dynamic temporal patterns showing high computational capabilities [8] [57]. Recurrent Neural Networks (RNNs) are dynamical systems characterized by feedback connections that can model reciprocal interactions between the neurons. Despite their great potential, these networks are difficult to train: their training results to be computationally expensive and slow [57]. In order to overcome these difficulties and exploit the advantages of RNNs, it has been studied a method of computing called Liquid State Machine (LSM), a method related to the Reservoir Computing (RC) model [57]. This model is composed of a Reservoir, the fixed recurrent structure, and a set of readouts, i.e., output neurons. The Reservoir uses spiking neuron models, e.g., Integrate and Fire (IF) [9] or Izhikevich [14]. An in depth analysis on LSMs will be developed in the following section. Recently this approach has got a great success in classification problems based on temporal data [8]. Since

these networks play a key role in many important safety-critical applications, their security represents a fundamental topic towards the future of the Machine Learning (ML) in a lot of applications in real life [21] [22]. Many adversarial attacks, adding human-imperceptible perturbations, have been studied to highlight the vulnerabilities of Deep Neural Networks (DNNs) [21] [22] [23] [24] [30]. Recently the work of Hazan and Manevitz [58] has analyzed the robustness in LSMs: they show that LSMs are not robust to particular damages of the neurons and to changes of the architecture. In our work, we test the vulnerability of LSNNs under imperceptible and robust adversarial examples, generated by the methodology explained in the Chapter 3.

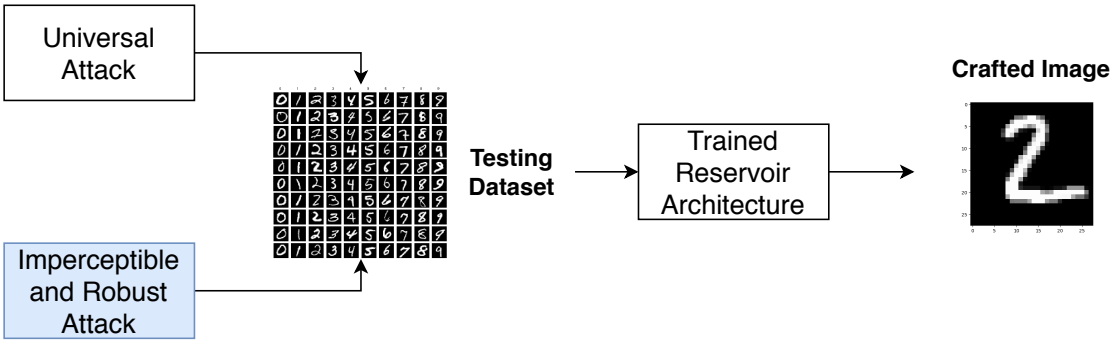


Figure 4.1: Overview of the content of our work.

4.1 Related Works

RNNs are dynamical systems in which the populations of neurons are linked by feedback connections, also present in biological brain modules. Since these connections allow to store the input informations in the internal state of the RNN, they contain a sort of dynamical memory and are capable to process informations based on temporal data [8]. The main problem of RNNs is related to their training: they are difficult to train using gradient descent methods, mainly for difficulties on convergence, dimensions of the network and number of the parameters [57]. The RC method solve these problems: they are composed of a random and fixed RNN, called Reservoir, and a set of output neurons, called readouts. The Reservoir is a structure that makes a temporal expansion of the input, mapping it into a high dimensional vector, representing the activities of the neurons. The readouts combine the signals coming from the Reservoir, usually in a linear way, and give the desired outputs. The concept of RC has been used for non-spiking networks and recently adopted for networks of spiking neurons: in this case, the method is called LSM. *In the LSM literature, the reservoir is often referred to as the liquid, following an intuitive metaphor of the excited states as ripples on the surface of a pool of water. Inputs to LSMs also*

usually consist of spike trains [57]. LSMs have been used in several applications [8]. Since the structure presents a division between the Reservoir and the readouts, the training is performed separately and many ideas have been proposed [57]. Recently, Hazan *et al.*, using the tool BindsNET [55], proposed a Reservoir model composed of a population of 625 LIF neurons. The input population from the image of a certain dataset is connected to the Reservoir. The outputs of the Reservoir, that creates a temporal representation of the data, are used to train, via gradient descent, a logistic regression model. Moreover, Hazan *et al.* [58] studied the vulnerability of LSMs, following the setup of Maas *et al.*: they observed, in contrast to Maas *et al.* [56], that LSMs are not robust to the noise.

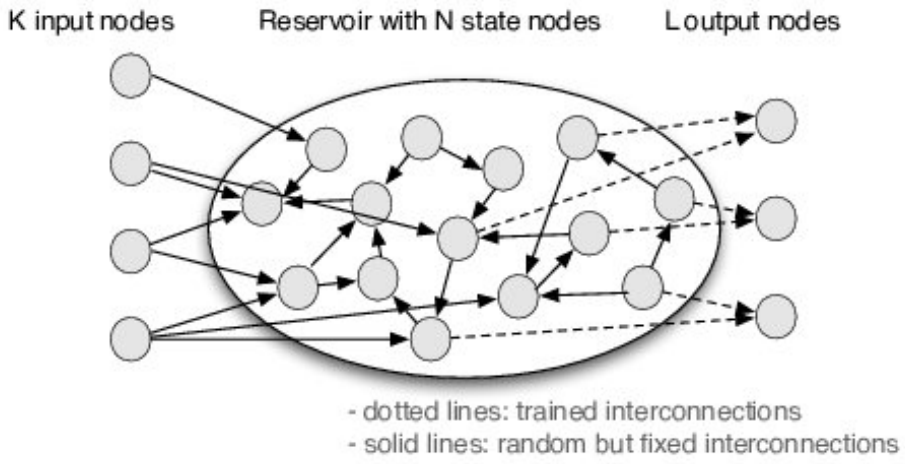


Figure 4.2: Schematic of the Reservoir Computing model. [Source: [59]]

4.2 Robustness of LSM under universal attacks

We consider as case of study the LSM implemented in BindsNET [55], described in the previous section. This network performs the classification of the MNIST dataset [34]: it is composed of 28*28 grayscale images representing handwritten digits, in particular 60.000 training images and 10.000 test images. The input pixels of each image are converted into Poisson spike trains [19]. The purpose of our work is to test this network under adversarial examples generated by our methodology. Since our algorithm modifies the pixels of a N*N search window, first of all we aim at analyzing the vulnerability of our LSM under fixed perturbations of different magnitudes applied to all the test images, performing the inference 112 times. Indeed, we consider 16 different windows (W) for each image, as shown in fig. 4.3: we compute

the test accuracy of the network applying noise of different magnitudes (N) for each window. The results are shown in table 4.1.

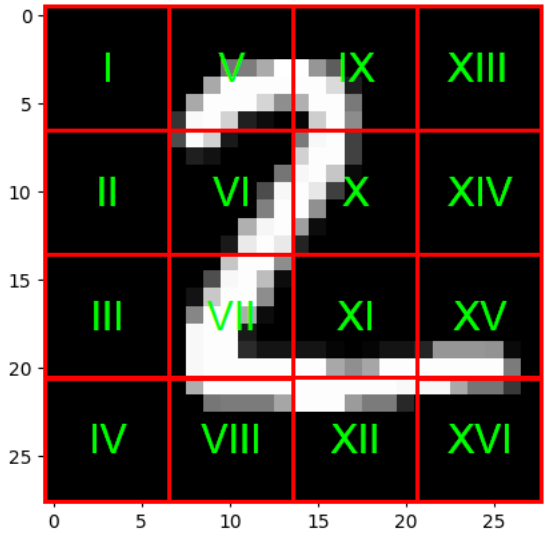


Figure 4.3: We consider 16 different 7*7 windows for each test images.

We can observe that the noise is more effective on the fifth window. It is important to underline that the starting accuracy, before the noise is added to the test images (N=0), results to be always different. The motivation of this apparently strange behavior is related to the Poisson encoding [19]: since this is a stochastic process, the probability of firing a spike is not always the same, even though the pixels do not change.

4.3 Evaluating our attack on LSM

4.3.1 Setup

We apply our methodology, explained in the Section 3.3.3 and showed in fig. 4.4, to the previously described LSM. Our black-box methodology can be applied to every kind of Neural Network because it modifies the pixels of the input images, without taking into account the parameters of the network. In this case, we choose a different metric for the distance $D(X^*, X)$. We express this value as the number of iterations that a sample needs to be misclassified. Moreover, we fix a value for D_{MAX}, that indicates a term of comparison for all the considered samples. Since the output

W/N	0	0.1	0.2	0.3	0.4	0.5	0.6
I	72.5	72.37	71.83	71.66	71.15	70.89	70.28
II	73.04	73.11	73.34	73.59	74.09	73.89	73.91
III	72.96	73.01	73.55	73.92	74	74.69	74.98
IV	72.71	73.2	72.83	72.84	72.77	72.84	72.78
V	72.82	72.38	71.9	70.94	70.16	69.24	**68.43**
VI	72.7	72.65	71.78	71.39	70.39	69.82	68.99
VII	72.6	72.5	72.53	71.89	71.89	71.51	71.36
VIII	72.81	72.68	72.59	72.64	72.55	71.88	71.69
IX	72.73	72.77	72.37	72.14	71.94	71.78	71.52
X	72.69	73.07	73.49	73.75	73.78	74.08	74
XI	72.58	72.44	71.96	71.44	71.52	71.03	70.59
XII	72.87	72.15	72.01	71.52	71.16	70.57	69.96
XIII	72.65	72.84	72.9	72.98	73.29	73.42	73.31
XIV	72.76	72.34	71.84	71.11	70.25	69.64	69.01
XV	72.77	72.83	72.54	72.79	72.72	72.32	72.16
XVI	72.56	72.97	73.24	73.41	73.56	73.66	73.93

Table 4.1: Accuracy of LSM when the noise of different magnitude (N) is added to each window (W).

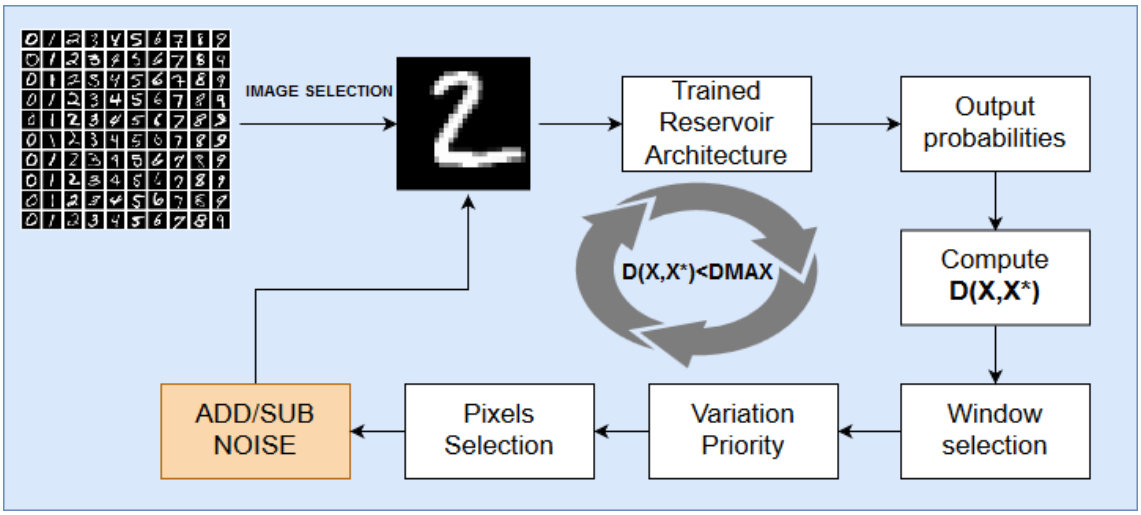


Figure 4.4: Our methodology to automatically generate adversarial examples.

probabilities of each sample comes from the Poisson encoding of the data, it can happen that a sample is misclassified, but at the next iteration it is classified correctly again. We evaluate the effect of our methodology on three examples, choosing as

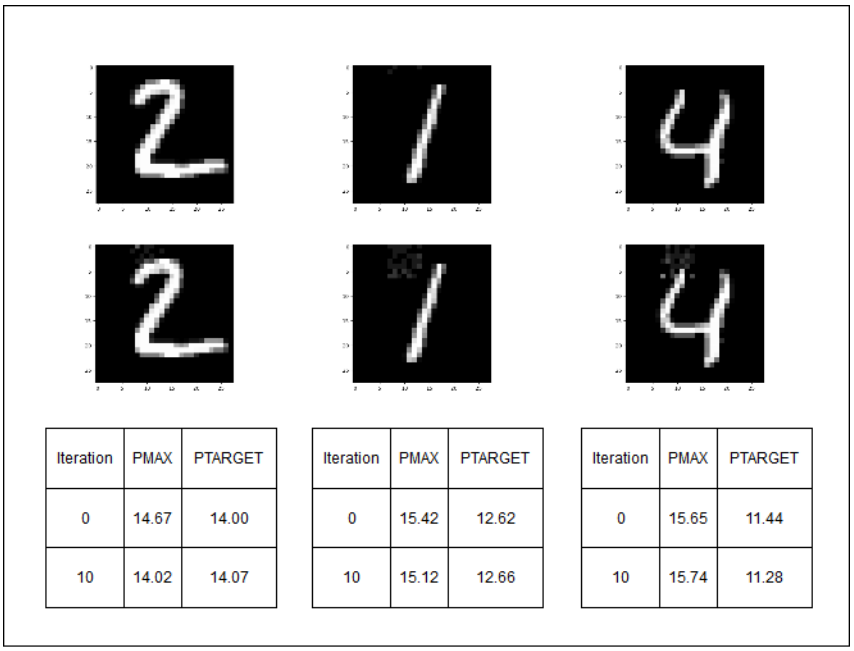
search window the fifth and the sixth windows, on which the perturbations are more effective.

4.3.2 Evaluation

We consider three different examples and we apply the attack generated by our methodology to both the chosen search windows, distinguishing two cases showed in fig. 4.5 and fig. 4.6. The target class for all the three examples is six, while the samples are classified by the network as two, one and four respectively. We choose a D_{MAX} equal to 10 iterations. In both the cases, we can observe that our attack works only on the first examples, because, after 10 iterations for the first case and 4 for the second case, the two is misclassified into a six. The noise is imperceptible to the human eye, but, also increasing the number of iterations, we cannot obtain a good robustness maintaining the imperceptibility in both the cases. This fact is probably due to the previously anayzed problem related to the Poisson encoding. Since it is a probabilistic process, the difference between the probabilities of the target and the maximum class does not decrease linearly. In the other examples, since the starting probabilities of the two considered classes are slightly higher compared to the first example, after 10 iterations we do not obtain a misclassification.

4.3.3 Open questions of future work

From the analyzed examples, we can deduce that it is difficult to draw generalizable conclusions for our LSM, since the generation of the spikes is based on the Poisson encoding process. From one side, in the examples having the starting probabilities not so far from each other, this process represents a weakness, because it results to be easy to fool the network. From the other side, when the starting gap is high enough, it represents a sort of antidote against imperceptible modifications of the pixels. Hence, we cannot say if this LSM is generally vulnerable or not to adversarial examples, but sure, in the most of the cases, the Poisson encoding process limits the performances of our black-box methodology. This behavior can be studied more in depth in future works, also considering the effect of a black-box attack not only to a single search window, but to many pixels selected between all the pixels of the images.

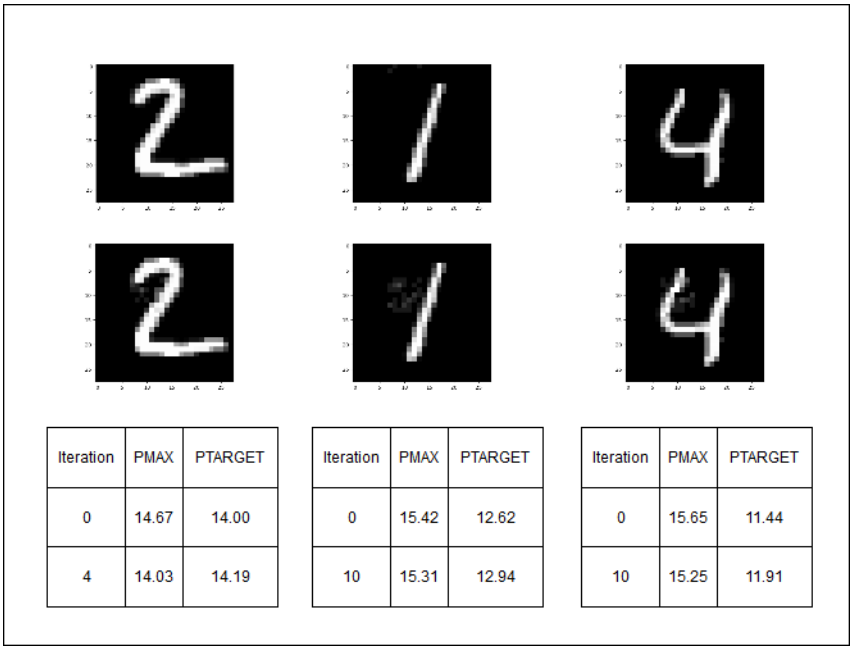


Iteration	PMAX	PTARGET
0	14.67	14.00
10	14.02	14.07

Iteration	PMAX	PTARGET
0	15.42	12.62
10	15.12	12.66

Iteration	PMAX	PTARGET
0	15.65	11.44
10	15.74	11.28

Figure 4.5: **Case 1** (V search window): our attack works only on the first example, but the attack does not result to be robust.



Iteration	PMAX	PTARGET
0	14.67	14.00
4	14.03	14.19

Iteration	PMAX	PTARGET
0	15.42	12.62
10	15.31	12.94

Iteration	PMAX	PTARGET
0	15.65	11.44
10	15.25	11.91

Figure 4.6: **Case 2** (VI search window): also in this case, the attack works only on the first example.

Chapter 5

CapsAttacks: Robust and Imperceptible Adversarial Attacks on Capsule Networks

Capsule Networks envision an innovative point of view about the representation of the objects in the brain and preserve the hierarchical spatial relationships between them. This type of networks exhibits a huge potential for several Machine Learning tasks like image classification, while outperforming Convolutional Neural Networks (CNNs). A large body of work has explored adversarial examples for CNNs, but their efficacy to Capsule Networks is not well explored. Despite the great success in the field of image classification, recent works [52] have demonstrated that, similarly to CNNs, CapsuleNets are also not immune to adversarial attacks. In this chapter, we target the following fundamental research questions:

1. If and how are the CapsuleNets vulnerable to adversarial examples?
2. How can adversarial attack for CapsuleNets be at the same time imperceptible and robust?
3. How does the vulnerability of CapsuleNets to adversarial attacks differ from that for the traditional CNNs?

To address these questions, we develop an algorithm to generate targeted imperceptible and robust (i.e., resistant to phisycal transformations) attacks considering a black-box scenario. To the best of our knowledge, we are the first to perform a comprehensive study of robustness/vulnerability of CapsuleNet to such adversarial attacks for the German Traffic Sign Recognition Benchmark [49], which is more crucial for autonomous vehicle related use cases. We also apply the same type of attacks to a 9-layer CNN having a similar starting accuracy, compared to the

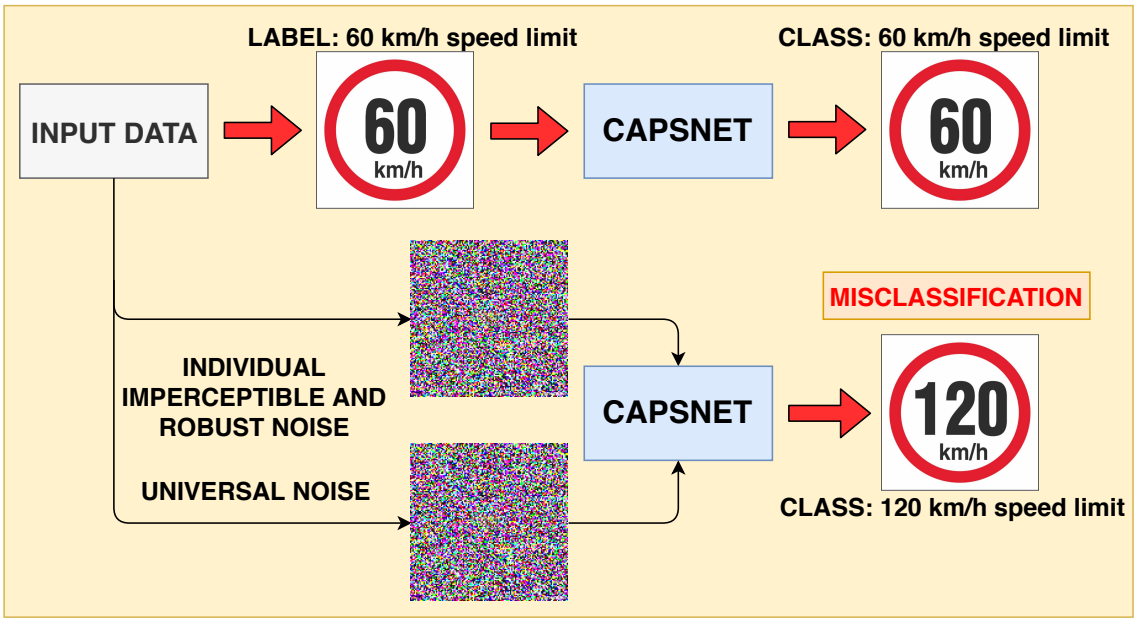


Figure 5.1: Overview of our work.

CapsuleNet. Our analyses show that CapsuleNets are more vulnerable than CNNs. Moreover, we investigate the impact of universal attacks on CapsuleNet with different (additive and subtractive) perturbations varying in their magnitudes. Our analyses show that, when the noise is subtracted from the intensity of the pixels, the accuracy of the network decreases quickly when compared to the other case. An overview of our approach is shown in Figure 5.1. **Our Novel Contributions:**

1. We analyze how the accuracy of CapsuleNet changes when universal attacks of different magnitudes of perturbation are added or subtracted to all the pixels of the input images of the GTSRB dataset.

2. We develop a novel methodology to automatically generate targeted imperceptible and robust adversarial examples.

3. We evaluate the robustness of CapsuleNet and a 9-layer CNN under the adversarial examples generated by our algorithm and under some affine transformation

5.1 Related Works

Capsules were first introduced by Hinton *et al.* [48]. They are multi-dimensional entities that are able to learn hierarchical information of the features. Compared to traditional CNNs, a CapsuleNet has the capsule as the basic element, instead of the

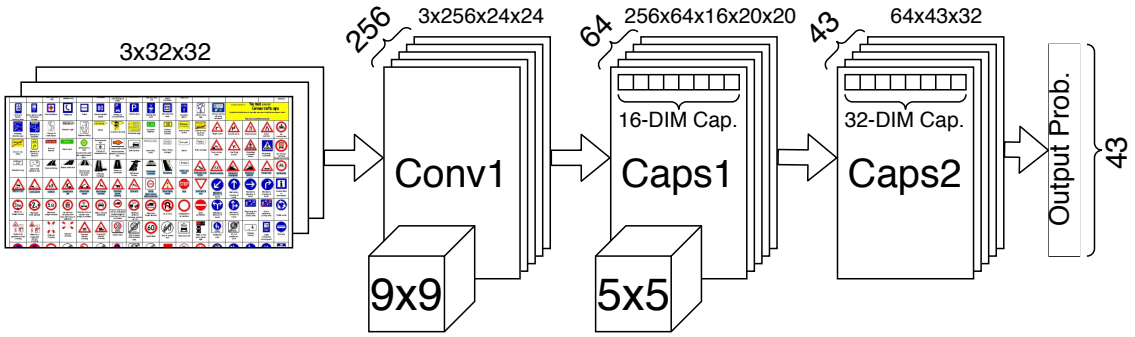


Figure 5.2: Architecture of the CapsuleNet for GTSRB dataset.

neuron. Recent researches about CapsuleNet architecture and training algorithms [4] [6] have shown competitive results, in terms of accuracy, for image classification task, compared to other state-of-the-art classifiers. Kumar *et al.* [50] proposed a CapsuleNet architecture, composed of 3 layers, which achieve good performance for the GTSRB dataset [49]. The architecture is visible in Figure 5.2. Recent works showed that CapsuleNet is vulnerable to adversarial attacks. Jaesik Yoon [52] analyzes how the accuracy of CapsuleNet changes applying Fast Gradient Sign Method (FGSM), Basic Iteration Method (BIM), Least-likely Class Method and Iterative Least-likely Class Method [22] to the MNIST dataset [34]. Frosst et al. [51] presented the technique called DARCCC (Detecting Adversaries by Reconstruction from Class Conditional Capsules), efficient on MNIST, Fashion-MNIST [53] and SVHN [54] datasets, to detect the crafted images.

5.2 Analysis: Evaluating the robustness of CapsuleNet

5.2.1 Experimental Setup

We consider the architecture of CapsuleNet, represented in Figure 5.2. It is composed of a convolutional layer, with kernel 9x9, a convolutional capsule layer, with kernel 5x5, and a fully connected capsule layer. We implement it in Tensorflow, to perform classification on the German Traffic Signs Dataset [49]. This dataset is composed of $32 \cdot 32$ RGB traffic signs images, divided into 34799 training examples and 12630 testing examples. The intensity of each pixel assumes a value from 0 to 1. The number of classes is 43.

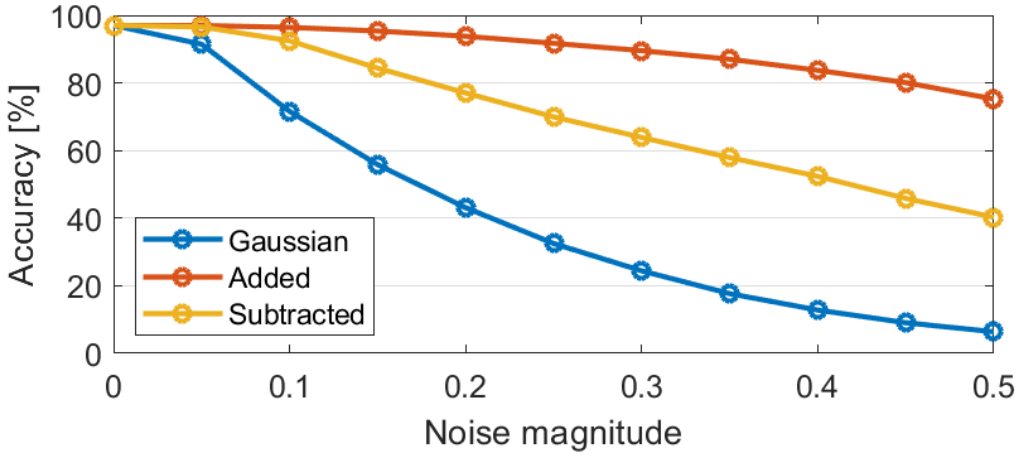
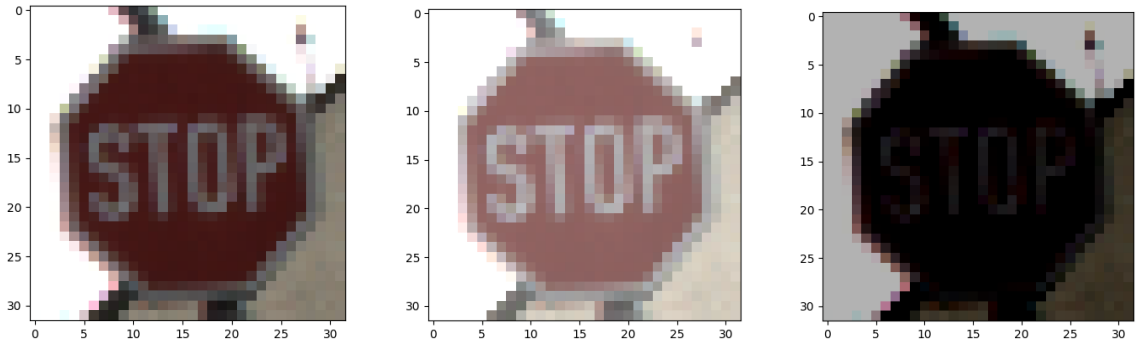


Figure 5.3: Addition and subtraction of fixed perturbations and addition of Gaussian perturbations.



(a) (b) (c)

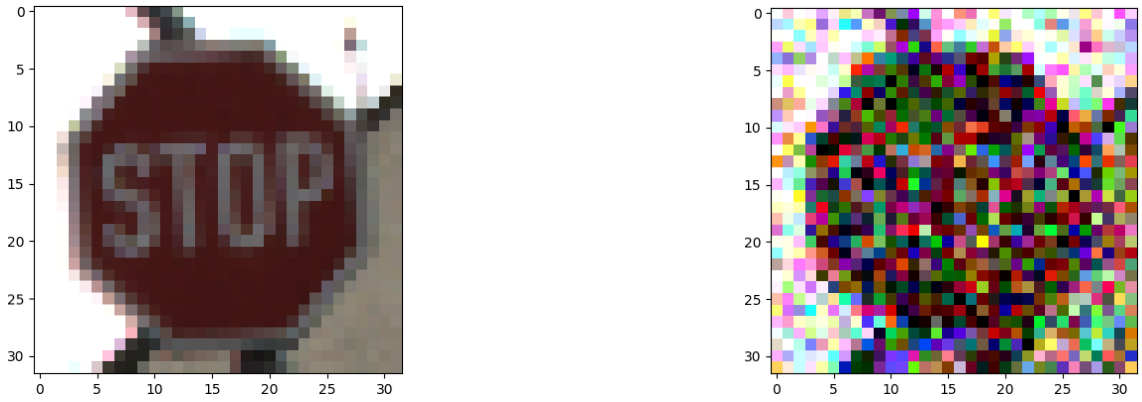
Figure 5.4: Comparison between images with fixed perturbations of magnitude 0.3 applied to all the pixels. (a) Without noise. (b) Added noise. (c) Subtracted noise.

5.2.2 Accuracy of the CapsuleNet under universal adversarial attacks

We analyze the accuracy of the CapsuleNet applying two different types of universal attacks to all the pixels of all the testing images:

- Addition and subtraction of fixed perturbations of different magnitudes.
- Addition of gaussian perturbations of different magnitudes.

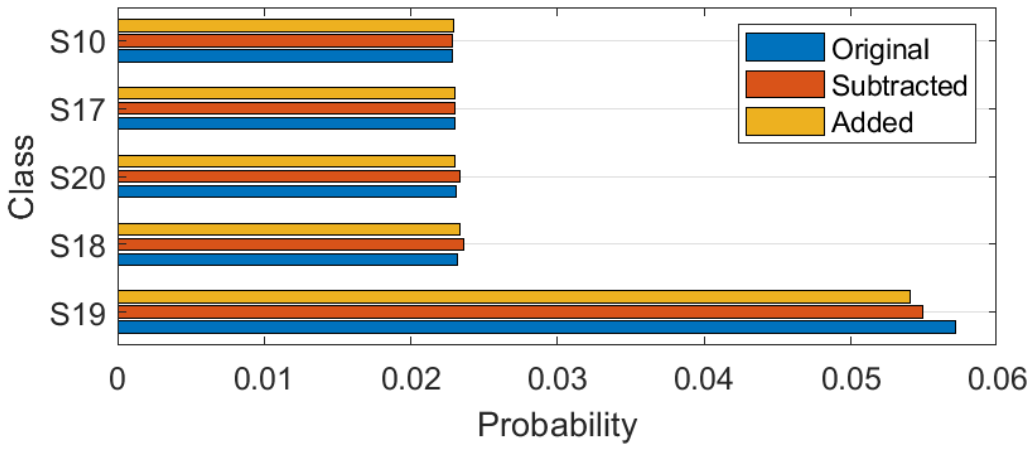
The accuracy of the CapsuleNet on the clean testing examples is approximatively 97%. Applying the noise with different magnitudes, we obtain the results in Figure 5.3. Examples of the resulting images under fixed perturbation and gaussian perturbation are shown in Figures 5.4 and 5.5, respectively. When the fixed noise is subtracted



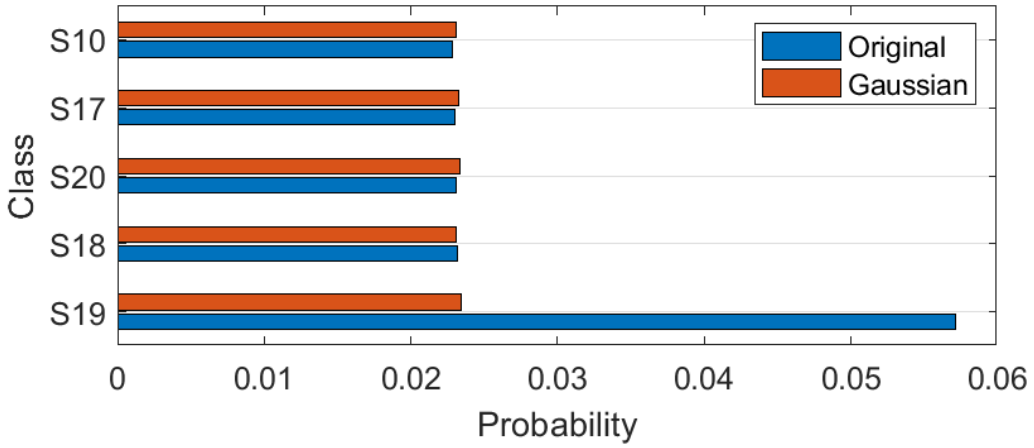
(a) (b)

Figure 5.5: Comparison between images with gaussian perturbations of magnitude 0.3 applied to all the pixels. (a) Without noise. (b) Added noise.

from the test images, the accuracy tends to decrease faster, compared to the case in which the noise is added to them. This effect can be explained by analyzing the test images in more detail. The average values of the pixel intensities of all the testing examples are equal to 0.33, 0.30 and 0.32 for the first, the second and the third RGB channel, respectively. *Hence, when the noise is subtracted, most of the pixels tend to become closer to 0 and the network easily misclassifies the image, because pixel values at the extremity of their range become meaningless.* As shown in the example of Figure 5.4, the image with subtracted noise (Figure 5.4c) is very dark, while the one with added noise (Figure 5.4b) is lighter than the original, but still recognizable also for humans. In Figure 5.3, the accuracy of the CapsuleNet with Gaussian noise decreases more sharply than the case relative to the fixed noise: in fact, the wide range of values of the Gaussian distribution, multiplied by the noise magnitude, create perceivable perturbations, as shown in Figure 5.5b. In our example, as represented in Figure 5.6, the sign is recognized as a "Stop" with probability 0.057. In this case, when the noise is added to the pixels, the probability decreases a little bit more than the case in which the noise is subtracted. In fact, in this example, the averages of the pixels are higher than the previous mean average, i.e., 0.52, 0.44 and 0.44, for each channel, respectively. Hence, the network easily misclassifies this image when the noise is added to the pixels. Nevertheless, the network correctly classifies our example because the gap between the probability of the "Stop" class and the highest among the other ones is large and a noise magnitude of 0.3 is not great enough to cause a misclassification. Otherwise, in the case of Gaussian noise, the probability of the "Stop" class decreases significantly, because the intensity of the noise is very perceivable, and thus decreases the quality of the image.



(a)



(b)

Figure 5.6: (a) Output probabilities in the case of fixed perturbations of magnitude 0.3 applied to all the pixels of our example. (b) Output probabilities in the case of Gaussian perturbations of magnitude 0.3 applied to all the pixels of our example.

5.3 Our Methodology: Automatic Generation of Targeted Imperceptible and Robust Adversarial Examples

An efficient adversarial attack is able to generate imperceptible and robust examples to fool the network. First we analyze the importance of these two concepts and then we describe our algorithm, based on them.

S1		Beware of ice/snow	S11		20 km/h speed limit
S2		Children crossing	S12		30 km/h speed limit
S3		Double curve	S13		100 km/h speed limit
S4		General caution	S14		Roads narrow on the right
S5		No passing	S15		Pedestrians
S6		Priority road	S16		Bicycle crossing
S7		60 km/h speed limit	S17		End of all speed and passing limits
S8		80 km/h speed limit	S18		Dangerous curve to the left
S9		120 km/h speed limit	S19		Stop
S10		Slippery road	S20		Wild animals crossing

Figure 5.7: All the considered traffic signs.

5.3.1 Imperceptibility and robustness of adversarial examples

An adversarial example can be defined *imperceptible* if the modifications of the original sample are so small that humans cannot notice them. To create an imperceptible adversarial example, we need to add the perturbations in the pixels of the image with the highest standard deviation. In fact the perturbations added in high variance zones are less evident and more difficult to detect with respect to the ones applied in low variance pixels. Considering an area of $M \cdot N$ pixels x, the standard deviation (SD) of the pixel $x_{i,j}$ can be computed as the square root of the variance as in

Equation 5.1, where μ is the average value of the $M \cdot N$ pixels:

$$SD(x_{i,j}) = \sqrt{\frac{\sum_{k=1}^{M}\sum_{l=1}^{N}(x_{k,l} - \mu)^2 - (x_{i,j} - \mu)^2}{M \cdot N}} \tag{5.1}$$

Hence, when the pixel is in a high variance region, its standard deviation is high and the probability to detect a modification of the pixel is low. In order to measure the imperceptibility, it is possible to define the distance between the original sample X and the adversarial sample X* as in Equation 5.2, where $\delta_{i,j}$ is the perturbation added to the pixel $x_{i,j}$:

$$D(X^*, X) = \sum_{i=1}^{M}\sum_{j=1}^{N}\frac{\delta_{i,j}}{SD(x_{i,j})} \tag{5.2}$$

This value indicates the total perturbation added to all the pixels under consideration. We define also D_{MAX} as the maximum total perturbation tolerated by the human eye. An adversarial example can be defined *robust* if the gap function, i.e., the difference between the probability of the target class and the maximum class probability is maximized:

$$GAP = P(target\,class) - max\{P(other\,classes)\} \tag{5.3}$$

If the gap function increases, the example becomes more robust, because the modifications of the probabilities caused by some image transformations (e.g., compression or resizing) tend to be less effective. Indeed, if the gap function is high, a variation of the probabilities could not be sufficient to achieve a misclassification.

5.3.2 Generation of the attacks

We propose a greedy algorithm that automatically generates targeted imperceptible and robust adversarial examples in a black-box scenario, i.e., we assume that the attacker has access to the input image and to the output probabilities vector, but not to the network model. The methodology is shown in Algorithm 2. The goal of our iterative algorithm is to modify the input image in order to maximize the gap function (*imperceptibility*) until the distance between the original and the adversarial example is under D_{MAX} (*robustness*).
Moreover, the algorithm takes in account the fact that every pixel is composed by three different values, since the images are based on three channels. Our algorithm chooses a subset P of pixels, included in the area of $M \cdot N$ pixels, with the highest SD for every channel, so that their possible modification is difficult to detect. Then,

Algorithm 2 : Our Methodology

Given: original sample X, maximum human perceptual distance D_{MAX}, noise magnitude δ, $M \cdot N$ pixels, target class, P, V
while $D(X^*, X) < D_{MAX}$ **do**
-Compute *Standard Deviation SD* for every pixel
-Select a subset P of pixels included in the area of $M \cdot N$ pixels with the highest *SD* for every channel
-Compute GAP, $GAP(-)$, $GAP(+)$
if $GAP(-) > GAP(+)$ **then**
$VariationPriority(x_{i,j}) = [GAP(-) - GAP] \cdot SD(x_{i,j})$
else
$VariationPriority(x_{i,j}) = [GAP(+) - GAP] \cdot SD(x_{i,j})$
end if
-Sort in descending order VariationPriority for every channel
-Select V pixels with highest VariationPriority between the three channels
if $GAP(-) > GAP(+)$ **then**
Subtract noise with magnitude δ from the pixel in the respective channel
else
Add noise with magnitude δ to the pixel in the respective channel
end if
-Compute $D(X^*, X)$ as the sum of the $D(X^*, X)$ of every channel
-Update the original example with the adversarial one
end while

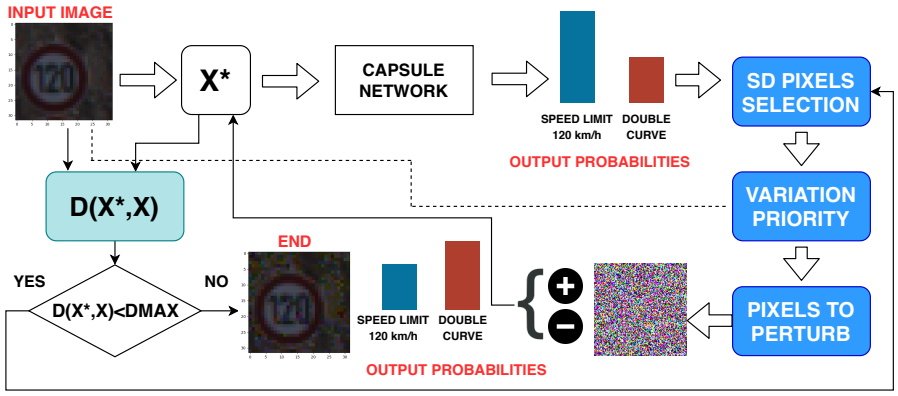


Figure 5.8: Overview of our methodology to generate adversarial examples.

the gap function is computed as the difference between the probability of the target class, chosen as the class with the second highest probability, and the maximum

output probability. Hence, for each pixel of P, we compute $GAP(+)$ and $GAP(-)$: these quantities correspond to the values of the gap function, estimated by adding and by subtracting, respectively, a perturbation unit to each pixel. These gaps are useful to decide if it is more effective to add or subtract the noise. For each pixel of P, we consider the greatest value between $GAP(+)$ and $GAP(-)$ to maximize the distance between the two probabilities. Therefore, for each pixel of P, we calculate the Variation Priority by multiplying the gap difference to the SD of the pixel. This quantity indicates the efficacy of the pixel perturbation. For every channel, the P values of Variation Priority are ordered and the highest V values, between the three channels, are perturbed. Then, starting from $3 \cdot P$ values of Variation Priority, only V values are perturbed. According to the highest value of the previous computed $GAP(+)$ and $GAP(-)$, the noise is added or subtracted. Once the original input image is replaced by the adversarial one, the next iteration can start. The iterations continue until the distance between the original and adversarial example overcomes D_{MAX}. The scheme of our algorithm is shown in Figure 5.8.

5.4 Impact of our attack on the CapsuleNet and the VGGNet

5.4.1 Experimental Setup

We apply our methodology, showed in Section 4.2, to the previously described CapsuleNet architecture and to a 9-layer VGGNet, implemented in Tensorflow and represented in Figure 5.9. The VGGNet, trained for 30 epochs, achieves a test accuracy equal to 97%. Therefore, it allows us to have a good comparison between the two networks, since their accuracy is very similar. To verify how our algorithm works, we test it on two different examples. We consider M=N=32, because the GTRSB dataset is composed by 32·32 images and P=100. The value of δ is equal to the 10% of the maximum value between all the pixels. The parameter D_{MAX} depends on the SD of the pixels of the input image: its value changes according to the different examples because $D(X^*, X)$ does not increase in the same way for each example.

5.4.2 Our methodology applied to the CapsuleNet

We test CapsuleNet on two different examples, shown in Figures 5.10a and 5.12a. For the first one, we distinguish two cases, in order to verify that our algorithm works independently from the target class, but with different final results:

- **Case I**: the target class is the class relative to the second highest probability between all the beginning output probabilities.

- **Case II**: the target class is the class relative to the fifth highest probability between all the beginning output probabilities.

We can make the following analyses on our examples:

1. CapsuleNet classifies the input image shown in Figures 5.10a and 5.11a as "120 km/h speed limit" with a probability equal to 0.0370.
   For the Case I, the target class is "Double curve" with a probability of 0.0297. After 8 iterations of our algorithm, the image (in Figure 5.10b) is classified as "Double curve" with a probability equal to 0.033. Hence, *the probability of the target class has overcome the initial one*, as shown in Figure 5.17a. The traffic signs relative to every class are showed in the table in Figure 5.7. At this step, the distance $D(X^*, X)$ is equal to 250.41. Increasing the number of the iterations, the robustness of the attack increases as well, because the gap between the two probabilities increases, but also the perceptibility of the noise becomes more evident. After the iteration 20, the distance grows above $D_{MAX} = 650$: the sample is represented in Figure 5.10c.

   For the Case II, the probability relative to the target class "Beware of ice/snow" is equal to 0.0249, as shown in Figure 5.17b. The gap between the maximum probability and the probability of the target class is higher than the gap in Case I. After 20 iterations, the network has not still misclassified the image (in Figure 5.11b). In Figure 5.17b we can observe that, however, the gap between the two classes has decreased a lot, until the iteration 22, when the value of the distance overcomes $D_{MAX} = 650$. In this case, we show that our algorithm still works, but, since we choose a target class which does not have the second highest probability, the initial gap between the two probabilities is high and the algorithm needs to perform several iterations to fool the network. Therefore, after many iteration, the noise becomes more perceivable.

2. CapsuleNet classifies the input image shown in Figure 5.12a as "Children crossing" with a probability equal to 0.042. The target class is "60 km/h speed limit" with a probability equal to 0.0331. After 20 iterations, the network does not misclassify the image (in Figure 5.12b) because the probability of the target class does not overcome the initial maximum probability, as shown in Figure 5.17c. The gap between the two probabilities increases until the iteration 32, when the sample is misclassified and the value of the distance overcomes $D_{MAX} = 146$. We can observe that the noise is perceivable, but only on the background of the image, as represented in Figure 5.12c.

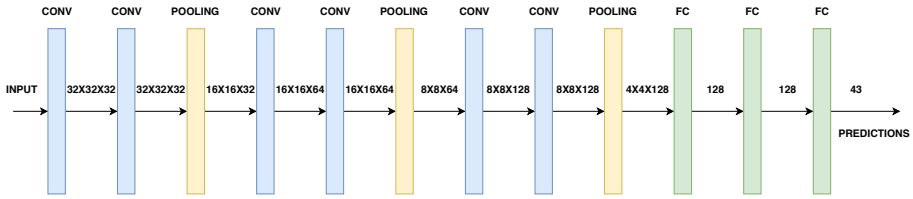
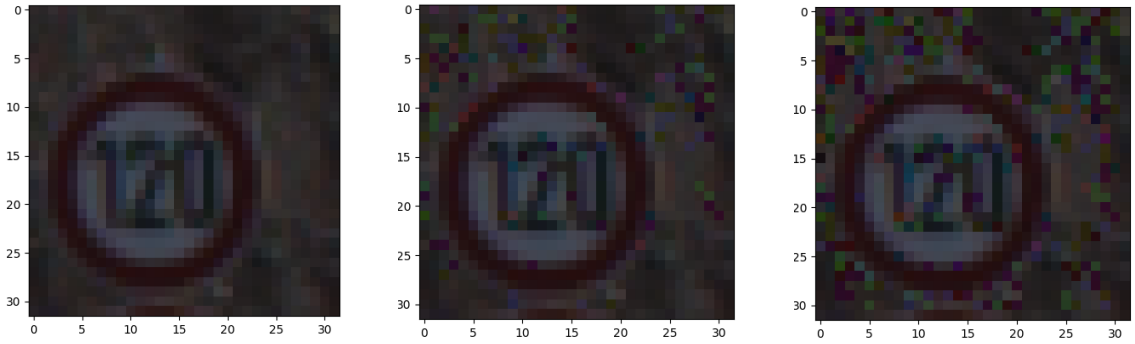
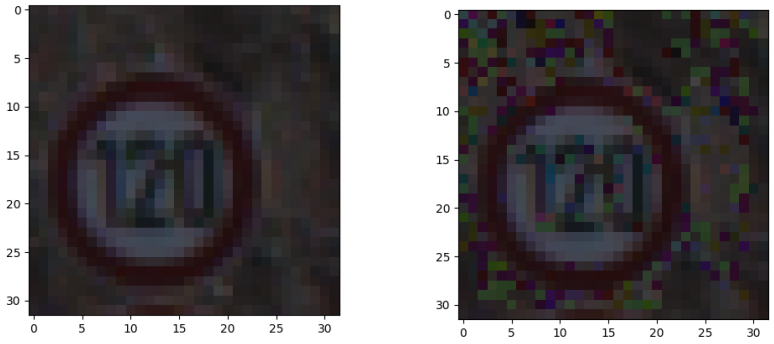


Figure 5.9: Architecture of the 9-layer VGGNet.



(a) (b) (c)

Figure 5.10: CapsuleNet images. (a) Original input image of Example 1. (b) Image misclassified by CapsuleNet at the iteration 8 for Case I. (c) Image misclassified by CapsuleNet at iteration 20 for Case I.

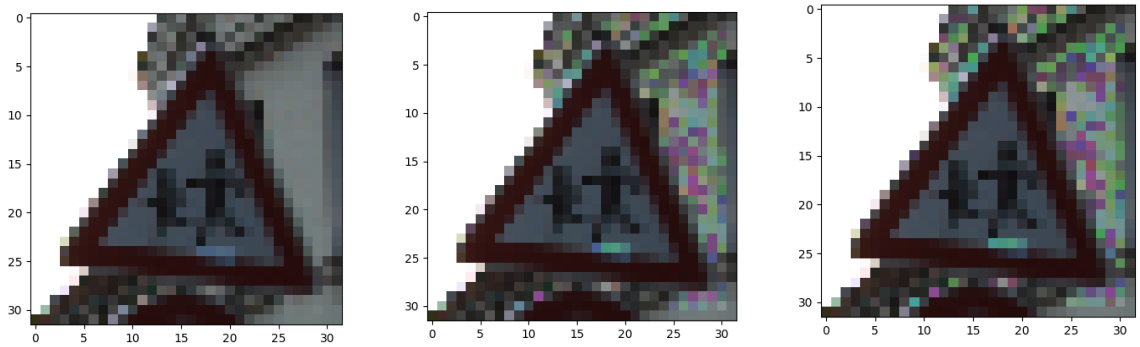


(a) (b)

Figure 5.11: CapsuleNet images (a) Original input image of Example 1. (b) Image at the iteration 20 for Case II

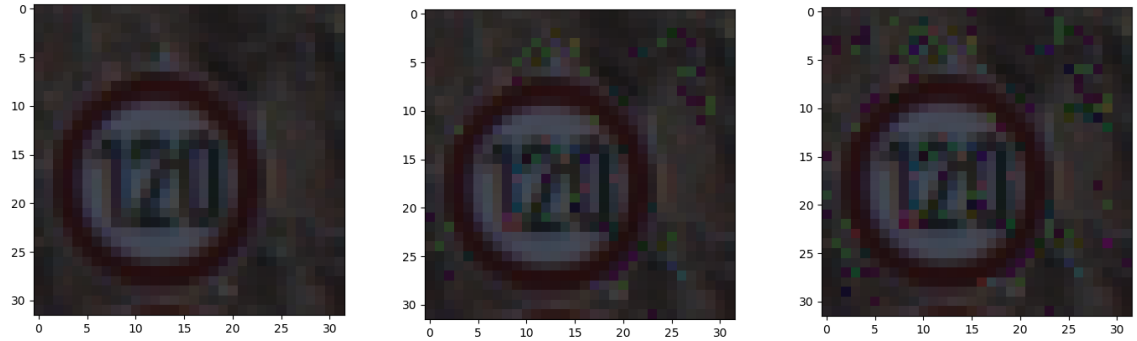
5.4.3 Our methodology applied to a 9-layer VGGNet

In order to compare the robustness of the CapsuleNet and the 9-layer VGGNet, we choose to evaluate the previous two examples, misclassified by the CapsuleNet. For the first example, we consider only the Case I as benchmark. The VGGNet classifies



(a) (b) (c)

Figure 5.12: CapsuleNet images (a) Original input image for Example 2. (b) Image at the iteration 20. (c) Image misclassified by CapsuleNet at the iteration 32.



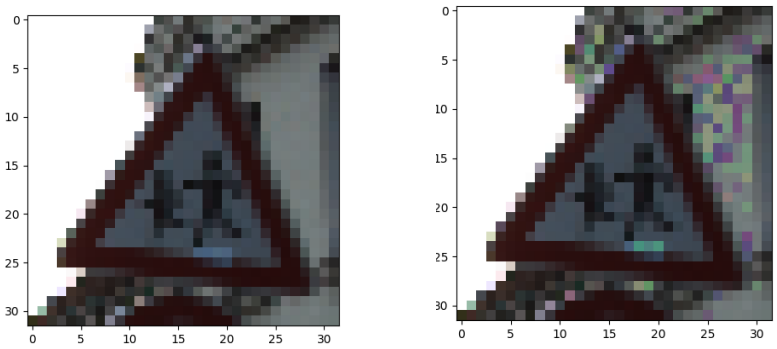
(a) (b) (c)

Figure 5.13: VGGNet images (a) Original input image for Example 1 (b) Image at the iteration 3. (c) Image at the iteration 15.

the input images with different output probabilities, compared to the ones obtained by the CapsuleNet. Therefore, our metric to evaluate how much VGGNet is resistant to our attack is based on the value of the gap at the same distance. We can make the following considerations on our two examples:

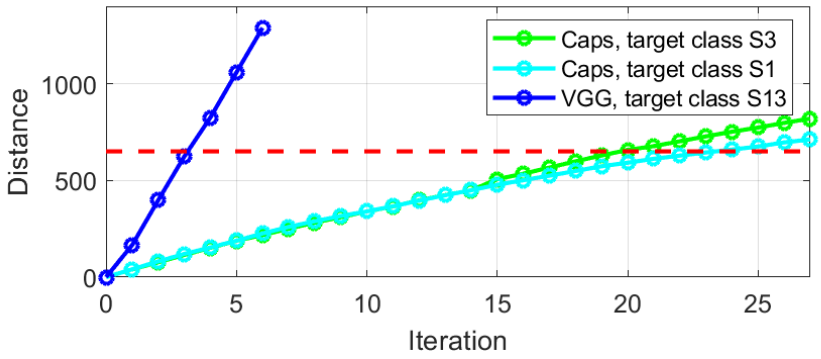
1. the VGGNet classifies the input image (in Figure 5.13a) as "120 km/h speed limit" with a probability equal to 0.979. The target class is "100 km/h speed limit" with a probability equal to 0.0075. After 3 iterations, the distance overcomes $D_{MAX} = 650$ and the network do not misclassify the image (in Figure 5.13b): our algorithm would needs to perform more iterations before obtaining a misclassification, since the two probabilities at the starting point were much distant, as shown in Figure 5.16a.

2. the VGGNet classifies the input image (in Figure 5.14a) as "Children crossing"

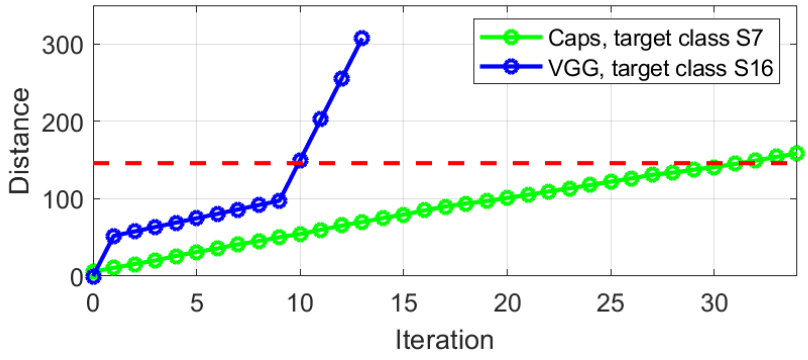


(a) (b)

Figure 5.14: VGGNet images (a) Original input images for Example 2. (b) Image at the iteration 9.



(a)

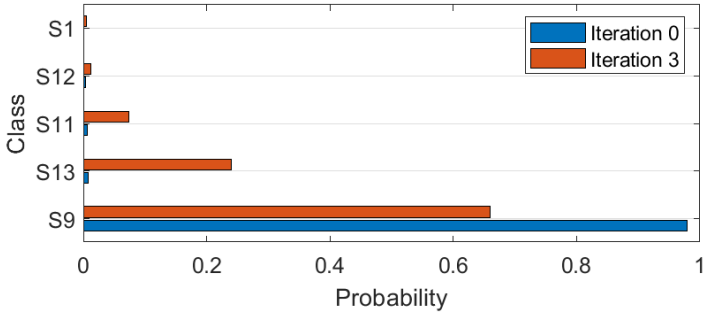


(b)

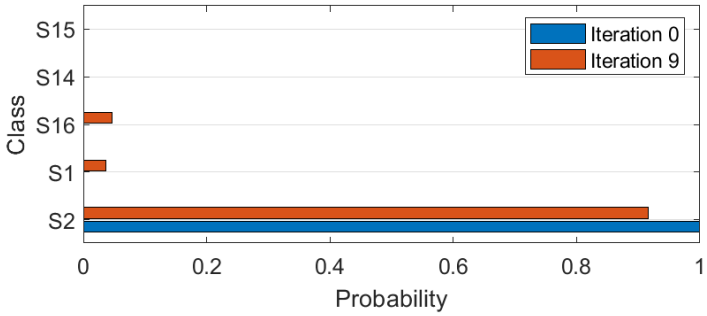
Figure 5.15: (a) $D(X^*, X)$ behavior for the Example 1. (b) $D(X^*, X)$ behavior for the Example 2.

with a probability equal to 0.99. We choose as target class "Bicycles crossing"

with the third highest probability, 0.0001. After 9 iterations, the distance overcomes $D_{MAX} = 146$ and the network does not misclassify the image (in Figure 5.14b). As in the previous case, this fact happens because the starting probabilities are very far, as shown in Figure 5.16b. In 5.13c, when $D(X^*, X)$ has overcome D_{MAX}, the noise is more perceivable than in the CapsuleNet at the same iteration (in Figure 5.10b).



(a)

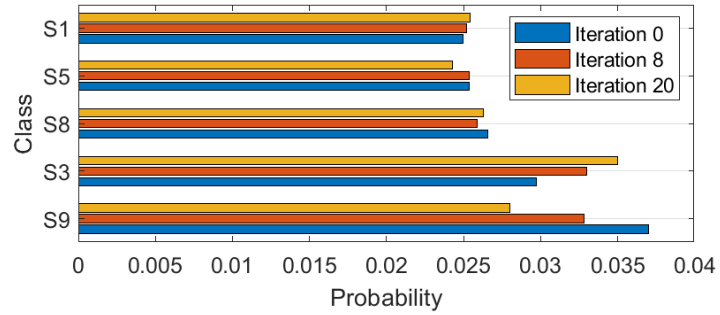


(b)

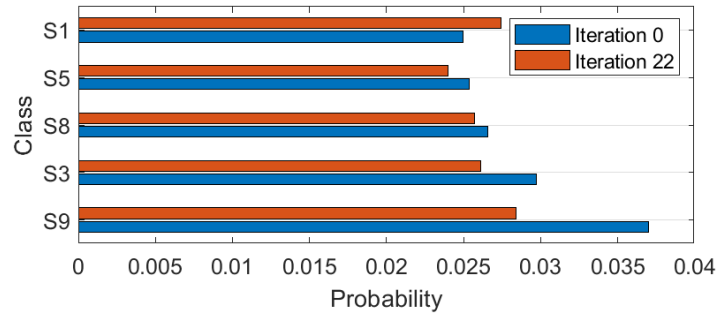
Figure 5.16: VGGNet results (a) Output probabilities of the Example 1: blue bars represent the starting probabilities and orange bars the probabilities at the D_{MAX}. (b) Output probabilities of the Example 2: blue bars represent the starting probabilities and orange bars the probabilities at the D_{MAX}.

5.4.4 Comparison and results

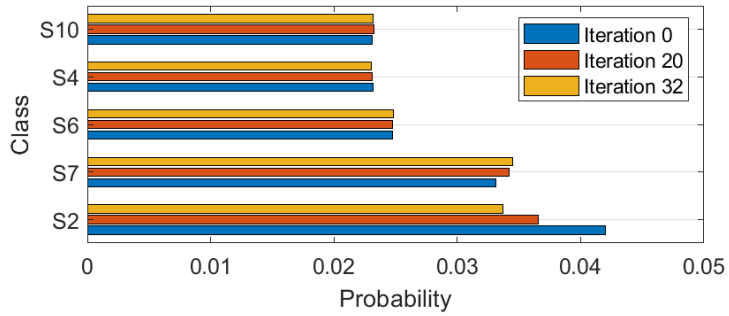
From our analyses, we can observe that the 9-layer VGGNet is more resistant to our adversarial attack compared to CapsuleNet, since the perturbations are less perceivable. Our consideration is justified from the graphs in Figure 5.15: in the cases of VGGNet, the value of $D(X^*, X)$ increases more sharply than for CapsuleNet. So the percipience of noise in the image can be measured as the value of $D(X^*, X)$ divided by the number of iterations: the noise in VGGNet becomes perceivable after



(a)



(b)



(c)

Figure 5.17: CapsuleNet results. (a) Output probabilities of the Example 1 - Case I: blue bars represent the starting probabilities, orange bars the probabilities at the point of misclassification and yellow bars at the D_{MAX}. (b) Output probabilities of the Example 1 - Case II: blue bars represent the starting probabilities and orange bars the probabilities at the D_{MAX}. (c) Output probabilities of the Example 2: blue bars represent the starting probabilities, orange bars the probabilities at the point of misclassification and yellow bars at the D_{MAX}.

few iterations. Moreover, we can observe that the choice of the target class plays an key role for the success of the attack. We notice that other features that evidence the differences between the VGGNet and the CapsuleNet were not considered. The

VGGNet is deeper and contains a larger number of weights, while the CapsNet can achieve a similar accuracy with a smaller footprint. This effect causes a disparity in the prediction confidence between the two networks. Thus, such observation raises an important research question: *is correct to compare the vulnerability of the CapsuleNet to a CNN with the same accuracy? And, if not, how can it be evaluated?* It is clear that the CapsuleNet has a much higher learning capability, compared to the VGGNet, but this phenomena has a negative drawback for the machine learning security point of view.

5.5 Analysis: vulnerability under affine transformations

In order to extend our analysis on the vulnerability of CapsNet under adversarial examples, we try to apply some affine transformation to the input images and observe how the predictions of CapsNet and the VGG change. We use three different types of transformations: rotation, shift and zoom.
As we can observe in fig. 5.18 and fig. 5.20, both the networks fail when the image is zoomed of x1.5 (in this case the probability of "Stop" for CapsNet is equal to 0.023, so near enough to the probability of "Road Work") and when it is shifted of four pixels into the top-left direction. In the examples in fig. 5.20, both the networks always fail when the image is zoomed of x1.5 and when it is rotated of 30°. Moreover, CapsNet also fails when the image is shifted of four pixels into the top-left direction. In the examples in fig. 5.21, CapsNet fails more times than the VGGNet: this is due to the fact that, in this image, the starting maximum probability is lower than the previously analyzed two cases. Hence, we can say that this behavior is consistent with the results obtained by the analysis performed in the Section 5.4. CapsNet and the VGGNet show a similar behavior under these affine transformations, but CapsNet results to be more vulnerable to them, mostly in the examples having a small gap between the first and the second highest probabilities.

5.6 Conclusions

In this work, we proposed a novel methodology to generate targeted adversarial attacks in a black box scenario. We applied our attack to the German Traffic Sign Recognition Benchmarks (GTSRB) dataset and we verified its impact on a CapsuleNet and a 9-layer VGGNet. Our experiments show that the VGGNet appears more robust to the attack, while the modifications of the pixels in the traffic signs are less perceivable when our algorithm is applied to the CapsuleNet. Moreover, also applying some affine transformations to the input images, we observed that

Figure 5.18: We apply some affine transformations to the original image, classified as "Stop" by both the networks.

the VGGNet results to be more robust than CapsNet. These results are consistent with the behavior of CapsNet observed applying our attack. We found an important weakness in the CapsuleNets: in an era in which the autonomous driving community is looking for high security of automatic systems in safety-critical environments, the CapsuleNet does not guarantee a sufficient robustness. Hence, further modifications of the CapsuleNet architecture need to be designed to reduce its vulnerability to adversarial attacks.

Figure 5.19: The signals in which the previous examples are incorrectly classified.

Figure 5.20: We apply some affine transformations to the original image, classified as "Speed limit (30 km/h)" by both the networks.

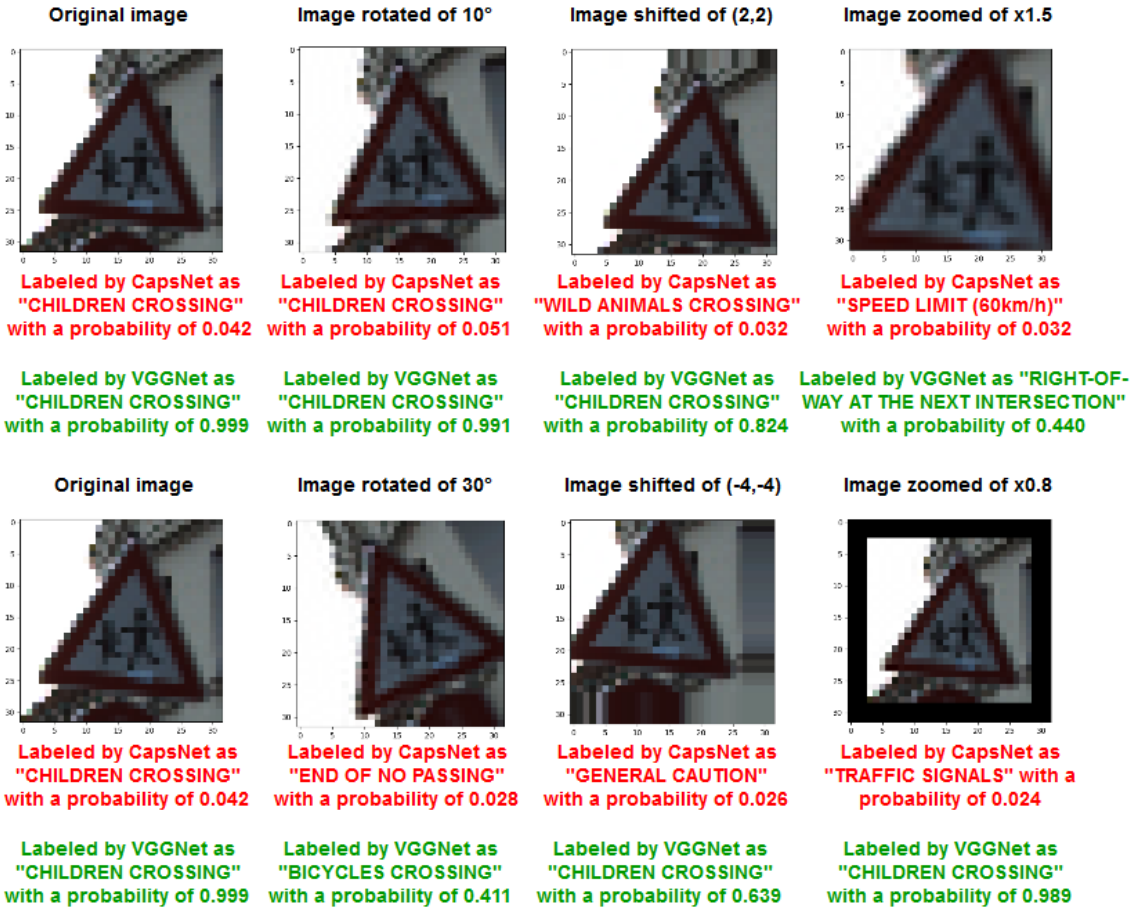


Figure 5.21: We apply some affine transformations to the original image, classified as "Children crossing" by both the networks.

Chapter 6

Conclusions

Adversarial examples have represented a fundamental topic in the scientific community about the analysis of the vulnerabilities of many Neural Networks in the last years. However, since the automation and many important applications using the Artificial Intelligence play today and will play tomorrow a primary role in our increasingly modern world, it becomes fundamental to study in depth the security of Neural Networks. The purpose of our work is to give an important contribution to the security in safety-critical environments, analyzing the effect of black-box imperceptible and robust perturbations. CapsuleNetworks and Spiking Neural Networks show a great success in several tasks and will play a key role in the future of Machine Learning: we highlight their potential weaknesses under a black-box methodology in order to improve these networks against dangerous attacks.

If you get up in the morning and think the future is going to be better, it is a bright day. Otherwise, it's not. (Elon Musk)

Bibliography

[1] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey.*Proceedings of the IEEE*, 105(12):22952329, 2017.

[2] A. Bhandare, M. Bhide, P. Gokhale, and R. Chandavarka. Applications of Convolutional Neural Networks, *In IJCSIT* ,2016.

[3] A. Canziani, A. Paszke, and E. Culurciello. An analysis ofdeep neural network models for practical applications. InIEEE International Symposium on Circuits and Systems, 2016.

[4] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. *In NIPS*,2017.

[5] R. Mukhometzianov, and J. Carrillo. CapsNet comparative performance evaluation for image classification. arXiv preprint arXiv 1805.11195,2018.

[6] G. E. Hinton, N. Frosst, and S. Sabour. Matrix capsules with em routing. *In ICLR*, 2018

[7] D. Lasitsa and A. Zhilenkov. Prospects for the Development and Application of Spiking Neural Networks. In *IEEE*, 2017.

[8] F. Ponulak and A. Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. In *ACTA Neurobiologiae experimentalis* 2011.

[9] Gerstner, W. and Kistler, W. K. (2002). Spiking Neuron Models. Cambridge University Press.

[10] W. Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. In *Neural Networks*, 1997

[11] Vreeken, J., Spiking neural networks, an introduction, Technical Report UU-CS-2003-008, Institute for Information and Computing Sciences, Utrecht University, 2002.

[12] Tavanaei A., Ghodrati M., Kheradpisheh S. R., Masquelier T., Maida A. S. (2018). Deep learning in spiking neural networks. *arXiv preprint arXiv:1804.08150*.

[13] S. Ye, S. Wang, X. Wang, B. Yuan, W. Wen and X. Lin. Defending DNN adversarial attacks with pruning and logits augmentation. In *ICLR*, 2018.

[14] E.M.Izhikevich, Simple Model of Spiking Neurons *IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 14, NO. 6, NOVEMBER 2003*

[15] E.M. Izhikevich, Which Model to use for Cortical Spiking Neurons?, *IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 15, NO. 5, SEPTEMBER 2004*

[16] M. Davies et al., Loihi: a Neuromorphic Manycore Processor with On-Chip Learning, *IEEE Micro, vol. 38, no. 1, 2018*

[17] D.O. Hebb, The Organization of Behavior, A NeuroPsychological Theory, Wiley, New York, 1949

[18] M. Fatahi, M. Ahmadi, M. Shahsavari, A. Ahmadi and P. Devienne. *evt_MNIST*: A spike based version of traditional MNIST. In *ICNRAECE*, 2016

[19] Heeger, Poisson model of spike generation, Handout, University of Standford, vol. 5, 2000.

[20] A. Shafahi, W. R. Huang, C. Studer, S. Feizi, and T. Goldstein. Are adversarial examples inevitable? *arXiv preprint arXiv:1809.02104*, 2018.

[21] I. Goodfellow, J. Shlens and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[22] A. Kukarin, I. Goodfellow and S. Bengio. Adversarial examples in the physical world. In *ICLR*, 2017.

[23] A. Arya, V. Ravi, V. Tejasviram, N. Sengupta and N. Kasabov. Cyber Fraud Detection using Evolving Spiking Neural Network. In *ICIIS*, 2016.

[24] J. Zhang and X. Jiang. Adversarial Examples: Opportunities and Challenges, *arXiv preprint arXiv:1809.04790*, 2018.

[25] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Celik, A. Swami . Practical Black-Box Attacks against Machine Learning. In *ACM*, 2017.

[26] Bo Luo, Yannan Liu, Lingxiao Wei, and Qiang Xu. Towards Imperceptible and Robust Adversarial Example Attacks against Neural Networks. *arXiv preprint arXiv:1801.04693*, 2018.

[27] R. Feinman, R. Curtin, S. Shintre and A. Gardner. Detecting adversarial examples from artifacts, *arXiv preprint arXiv:1703.00410*, 2017.

[28] A. Bhagoji, D. Cullina and P. Mittal. Dimensionality reduction as a Defense against Evasion Attacks on Machine Learning Classifiers, *arXiv preprint arXiv:1704.02654*, 2017.

[29] A. Bhagoji, D. Cullina, C. Sitawarin and P. Mittal. Enhancing Robustness of Machine Learning Systems via Data Transformations, *arXiv preprint arXiv:1704.02654*, 2017.

[30] Szegedy, C., Zaremba, W., and Sutskever, I. Intriguing properties of neural networks. *ICLR*, 2014.

[31] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *arXiv preprint arXiv:1712.07107*, 2017.

[32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks.*arXiv preprint arXiv:1706.06083*,2017.

[33] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacksagainst black-box machine learning models.arXiv preprint arXiv:1712.04248, 2017.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.

[35] N. Carlini, P. Mishra, T.Vaidya, Y Zhang, M.Sherr, C. Shields, D. Wagner, and W. Zhou. Hidden voice commands. In *USENIX Security Symposium*, 2016.

[36] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu. Dolphinatack: Inaudible voice commands. *arXiv preprint arXiv:1708.09537*, 2017.

[37] F.Khalid, M. Hanif, S. Rehman, J. Qadir, M. Shafique. FAdeML: Understanding the Impact of Pre-Processing Noise Filtering on Adversarial Machine Learning, *arXiv preprint arXiv:1811.01444*, 2018.

[38] P. O'Connor, D. Neil, S.Liu , T. Delbruck and M.Pfeiffer: Real-time classification and sensor fusion with a spiking deep belief network. In *Frontiers in Neuroscience*, 2013.

[39] F.Khalid, M. Hanif, S. Rehman, J. Qadir, M. Shafique. FAdeML: Understanding the Impact of Pre-Processing Noise Filtering on Adversarial Machine Learning, *arXiv preprint arXiv:1811.01444*, 2018.

[40] L. Holmstrom and P. Koistinen. Using additive noise in Back-Propagation Training. In *IEEE transactions on Neural Networks*, 1992.

[41] G. Hinton, S. Osindero and Y. Teh. A fast learning algorithm in Deep Belief Nets. In *Neural computation*, 2016.

[42] T. Tieleman and G. Hinton. Using fast weights to improve Persistent Contrastive Divergence. In *ICML*, 2009.

[43] E. Merino., F. Castrillejo, J. Pin, D. Prats. Weighted Contrastive Divergence, *arXiv preprint arXiv:1801.02567*, 2018.

[44] Goh, H., Thome, N., and Cord, M. (2010). Biasing restricted Boltzmann machines to manipulate latent selectivity and sparsity, in *NIPS workshop on deep learning and unsupervised feature learning*, (Whistler, BC)

[45] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle. Greedy Layer-Wise Training of Deep Networks. In *NIPS*, 2006.

[46] G. Hinton and R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networkss. In *Science*, 2006.

[47] A. Siegert. On the First Passage Time Probability Problem. *Physical Review*, 1951.

[48] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *ICANN*, 2011.

[49] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *IJCNN*, 2013.

[50] A. D. Kumar, R. Karthika, and L. Parameswaran. Novel deep learning model for traffic sign detection using capsule networks. *arXiv preprint arXiv:1805.04424*, 2018.

[51] N. Frosst, S. Sabour, G. Hinton. DARCCC: Detecting Adversaries by Reconstruction from Class Conditional Capsules, *arXiv preprint arXiv:1811.06969*, 2018.

[52] Adversarial Attack to Capsule Networks project. Available online at: https://github.com/jaesik817/adv_attack_capsnet.

[53] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. In *CoRR abs/1708.07747*, 2017.

[54] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[55] Hazan H, Saunders DJ, Khan H, Patel D, Sanghavi DT, Siegelmann HT and Kozma R (2018)BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python.Front.Neuroinform. 12:89. doi: 10.3389/fninf.2018.00089.

[56] Maass, W., Natschlger, T., and Markram, H. (2002). Computational models for generic cortical microcircuits. In J. Feng (Ed.), Computational neuroscience: A comprehensive approach. CRC-Press. Retrieved from papers/lsm-feng-chapter-149.pdf.

[57] Lukosevicius, M., and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. Computer Science Review, 3(3), 127149. doi:10.1016/ j.cosrev.2009.03.005.

[58] Hazan, H. and L. M. Manevitz (2012, Feb.). Topological constraints and robustness inliquid state machines.Expert Syst. Appl. 39(2), 15971606.

[59] Wyffels, Schrauwen and Stroobandt, System modeling with Reservoir Computing, 2018