# Steganalysis and steganography by deep learning

Mehdi Yedroudj

# THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

**En Informatique**

**École doctorale : Information, Structures, Systèmes**

**Unité de recherche : LIRMM**

# Steganalysis and Steganography by Deep Learning


Présentée par **Mehdi YEDROUDJ**


**Le 26/11/2019**

Sous la direction de **Marc CHAUMONT** et **Fréderic COMBY**

**Devant le jury composé de**

| | | | |
|---|---|---|---|
| Patrick BAS | DR-CNRS | Ecole centrale de Lille | Rapporteur |
| Caroline FONTAINE | DR-CNRS | ENS Paris-Saclay | Rapportrice |
| Sandra BRINGAY | PR | Université Paul Valéry Montpellier III | Présidente / Examinatrice |
| Rémi COGRANNE | MCF | Université de Technologie Troyes | Examinateur |
| Marc CHAUMONT | MCF-HDR | Université de Nîmes | Directeur de thèse |
| Frédéric COMBY | MCF | Université de Montpellier | Encadrant |

UNIVERSITÉ DE MONTPELLIER

# Acknowledgement

*I am grateful for Almighty Allah for giving me the strength and ability to fulfil this goal.*

*My sincere appreciation and gratitude go to my primary supervisor Doctor Marc CHAUMONT and my co-supervisor Doctor Frederic COMBY. Thank you for your help, advises and your availability, which have been essential for me to understand and adapt to the challenges of research. Your many reviews, corrections and constructive feedback of my work, especially those of this thesis, have been very important for their final qualities. Working under your direction has been very pleasant and for that, thank you.*

*Also, I would like to thank my examination committee: Professor Patrick BAS, Professor Caroline FONTAINE, Professor Sandra BRINGAY and Doctor Rémi COGRANNE for their time and efforts.*

*I dedicate great thanks with love to my beautiful family, My mother, my father, my brother and my precious sisters. I could not have achieved this without your prayers and your encouragements. Thank you for being by my side all the time.*

*My friends, it is a blessing to have you in my life. Many thanks are also due to the administration staff in LIRMM laboratory, and all the members of the ICAR team for being such kind and cooperative.*

*Finally, I would like to thank the Algerian Ministry of Higher Education and scientific research, and Constantine 2 University for funding my PhD thesis.*

# Contents

# List of Figures

# List of Tables

# Preface and positioning of the thesis

Since the 1990s, the Internet and storage devices have become more important in our daily lives. Nowadays, the Internet/storage devices (communication channels) offers an open space for information exchange, as it is an easy and perfect carrier. The information may consist in images, videos or audio. Such media may be used to conceal secret information, which is then transmitted over the communication channel. However, communicating secret information on a public channel, such as the Internet, can be dangerous as privacy and confidentiality are not guaranteed. Under such circumstances, it is important to find a way to send or receive information *from* and *to* specific persons secretly, i.e. without any suspicious from other persons. The communication on such a channel is known as *secret communication*. The latter has been an active field of research for several decades and flourished in the early 21st century.

The confidentiality and security of communications over an unsecured or monitored communication channel is associated with cryptography, which transforms a given message into an incomprehensible text. The encrypted text is transmitted on a public channel, where only the authorized person, who held an encryption key, can decrypt it. As for the stealthy communication, i.e. secret communication, it is associated with steganography. It strives to hide the very fact of the existence of communication. To further explain, in steganography, the encrypted message must be hidden insight of regular traffic, and only the associate person is aware of

its existence. With the appropriate key called *stego-key*, a hidden message can be retrieved and eventually decrypted.

Nowadays, there are hundreds of steganographic tools capable of hiding information within another innocent file. These innocent files are usually digital multimedia files known as *cover*. Speaking of images -the kind of files that we will be using in this thesis-, an image is a set of values (pixels) that can be used to embed a message. These digital media (images) are a perfect choice for steganography, as they are not suspicious (given the large number of images on the Internet). Furthermore, they are easy to process.

For modern steganography and more precisely adaptive steganography, the embedding process begins with the attribution of a cost to each pixel of the image. These costs reflect the level of detectability if modifying the pixel. Pixels with low modification cost are preferably modified at first to perfectly embed the message bits while minimizing a cover distortion measure. Modern steganography techniques (adaptive) start by calculating a cost map using a distortion function that serves the purpose of identifying pixels with low impact on the cover distortion. The way such a cost map is constructed is a crucial element in the designing of steganographic algorithms.

During the embedding process, steganographic methods (approach by modification) inevitably modify the statistical characteristics of images. Such non-natural distortions may be detected and reveal that secret communication is taking place. The techniques that attempt to detect the presence of such non-natural distortions to distinguish original content (cover) from media with a hidden message (known as stego) are called *steganalysis*. Steganalysis is considered as a hypothesis-testing problem since a steganalyst needs to determine whether or not the given digital medium is "stego". It is practically viewed as a classification problem and addressed using machine learning tools and more recently Deep Learning tools. The most commonly used tools in the steganalysis community are Ensemble classifiers [Kodovský et al., 2012], regularised linear classifier [Cogranne et al., 2015] and Convolutional Neural Networks [Chaumont, 2020].

Until 2015, we were talking about conventional steganalysis based on two-step machine learning approaches. First, a set of features is extracted from a database made of cover/stego images. Then, a given classifier is trained to discriminate, the most precise possible, the two classes of objects based on the part of the extracted features. Its accuracy is then evaluated on the remaining part. A well-extracted feature space can capture different dependencies between image elements (pixels, coefficient); these dependencies are often distorted during the message embedding. After 2015, and with the evolution of steganography field, more adapted and complex feature space was needed in steganalysis field. This led to the arrival of Deep Learning-Based steganalysis approach where the two-steps learning are combined into one-step learning. Thus the feature space is automatically explored, explaining why deep learning-based approach outperforms conventional methods.

Once again, the evolution of steganalysis has led the research community to work on new steganographic methods that can withstand the novel and well-designed steganalyst models. For that purpose, the researchers propose more and more complex and sophisticated distortion functions to obtain more refined cost-map. They even go further and propose a novel conception of steganography based on deep learning techniques. The new steganographic system appeared in 2018. In contrast, as steganalysis field is now challenged by this new generation of steganographic algorithms, steganalysis should then find and propose new image descriptors or new models that can cope with these updated steganographic systems.

The never-ending battle between steganography and steganalysis creates a self-stimulating environment that is beneficial for both fields. It is in this essence that we began this thesis.

The objective of this thesis was first to design a new, efficient and powerful framework for the steganalysis of digital images and more precisely uncompressed grey-scale images. Secondly, the proposed framework for steganalysis was incorporated to design a steganographic system based on deep learning where the entire embedding process is carried out automatically and without any human intervention to simulate embedding algorithms that use human-based rules.

This manuscript is composed of two parts: the first part devoted to the state of the art where we introduce the concepts and basic tools that will be necessary to understand the rest of the thesis, then the second part is devoted to the main contributions of this thesis.

In chapter 1, we provide an introduction to the two fields of steganography and steganalysis by reviewing the historical background of steganography and outlining the role of steganalysis and its different types that are described in the Simmons model. Then we refer to some recent cases of modern steganography used for legitimate and malicious purposes. The notations used in the thesis are introduced at the end of this section.

In chapter 2, we discuss machine learning concepts. We start by presenting what machine learning is while providing some basic concepts and definitions. Next, we explain the difference between two-step learning-based approaches and one-step learning-based approaches. After that, we will discuss the different types of learning algorithms for machine learning. We will mainly focus on feature extraction and classification techniques. Next, we review deep learning models and, more specifically, Convolutional Neural Networks (CNN). In the end, we conclude with the generative model, and we will present the all-new technique called Generative Adversarial Network (GAN).

Chapter 3 is devoted to steganography algorithms. We start by presenting steganography by mentioning the three steganography models defined by Simmons [Simmons, 1984]. Then we give the different philosophies of steganography. Next, we discuss modern steganography algorithms known as adaptive-steganography. The elementary building blocks of adaptive modern steganography algorithms are presented and discussed one by one, although we will mainly focus on the notion of a cost map (generated using a cost function known as distortion function). We conclude this chapter, presenting some famous state-of-the-art steganography algorithms.

Thanks to the emergence of deep learning and GAN models, new steganography approaches have appeared. It is known as *strategic adaptive steganography*. In

chapter 4, we discuss the *strategic adaptive steganography*, and we present the four families that include techniques and methods coming from this new approach of steganography.

Chapter 5 is dedicated to digital image steganalysis using machine learning techniques. The first sections will address the elementary notions of steganalysis for a better understanding. The following sections will be devoted to the essential primitives of steganalysis, whether two-step steganalysis methods (residue calculation, classification) or one-step steganalysis methods (CNN design). We conclude by illustrating some CNNs models designed for steganalysis ends. These CNN represent the state-of-the-art of nowadays steganalysis.

In chapter 6, we present our proposed framework, which is a deep learning-based approach mainly designed for spatial steganalysis purposes. The proposed model called Yedroudj-Net is a CNN that defeats the state-of-the-art in a classical clairvoyant scenario. We discuss in this section the strong part of the proposed CNN, as we put it to compete against other CNN models.

We then explore the ways to improve the performances of our proposed framework Yedroudj-Net. To this end, we have studied the effects of base augmentation on the performance of steganalysis using a CNN. In chapter 7, we present the results of this study using various experimental protocols and various databases to define the good practices in base augmentation for steganalysis.

In chapter 8, we propose a new fashion to do steganography, i.e. *strategic adaptive steganography* using Generative Adversarial Network. The proposed stenographic system competes against our proposed steganalysis model. For further details, we present the concept of strategic adaptive steganography while illustrating the three architectures models that we have proposed for steganographic purposes.

The thesis is concluded in Chapter 9. In this chapter, we give a general conclusion where we summarize the contributions of this thesis and outline the future directions and considered perspectives.

# Chapter 1

# Introduction

## Contents

The evolution of computer technology and telecommunications have contributed to a multitude of problems related to the protection (security) of information. This led to a broad scientific and technological development in order to respond to these new emerging challenges. Among these challenges, we find computer viruses, access authorization, copyright protection, and data integrity verification. In addition, it is possible to add issues such as authentication, conditional access, watermarking, digital signature, secret communication and steganography. In our work, we focus on the issue of secret communication, based mainly on the steganography.

We will discuss in Section. 1.1 the historical context of steganography and give examples of some ancient steganographic techniques. Then in Section. 1.2 we will discuss some uses of the nowadays steganography (modern steganography).

In Section. 1.3, we will present the prisoners' problem where we discuss the goal and role of each of the three actors of a steganographic system (Alice, Bob, and Eve).

The notation used throughout this dissertation is introduced in Section. 1.4.

## 1.1 A brief history of Steganography

If cryptography is the art of secrecy, then steganography may be considered as the art of hiding data secretly.To take a metaphor, steganography would be to make a file in your computer invisible where cryptography would be to lock this file - but there is nothing to stop you from combining the two techniques, just as you can lock it and then make it invisible.

Steganography has roots in Greek civilization with the word steganos, which means protected, or covered, and graphei which means writing, and translated as "covered writing." Steganography's purpose is to hide a message within an innocuous carrier called cover objects; this is to make the message imperceptible to anyone other than the legitimate recipient. The cover object may be basically anything, from

FIGURE 1.1: Illustration of a hidden message on the head of a person.

a physical text document to a painting, or even a piece of wood, as long as it can be used to deliver a hidden message to the intended recipient without raising suspicion of untrusted parties.

The emergence of steganography, the art of concealing information, is very old, and almost contemporary with that of cryptography. The first example dates back to the 5th century BC and is told in the biography of Herodotus [Fridrich, 2009]. The Greeks shaved a slave's hair, then tattooed a message on his head as shown in the figure 1.1. Once the hair was pushed back, the slave could cross enemy territory without arousing suspicion. Once there, all it took was another shave of the head to get the message back.

Since then, many examples of setganography have been reported over the past 2000 years.

Another example is *Invisible inks* [Kipper, 2003]. It has been the most widely used steganography method over the centuries. In the middle of the texts written in ink, a message is written using lemon juice, milk or certain chemicals, making the message disappear after the paper dries. It is invisible to the eye, but a simple flame, or a bath in a chemical reagent, reveals the message. Figure. 1.2 illustrates a good example of this technique.

The Second World War saw many forms of steganography [Guillot, 2013]. The methods were sometimes quite rudimentary, like this message sent by a German spy:

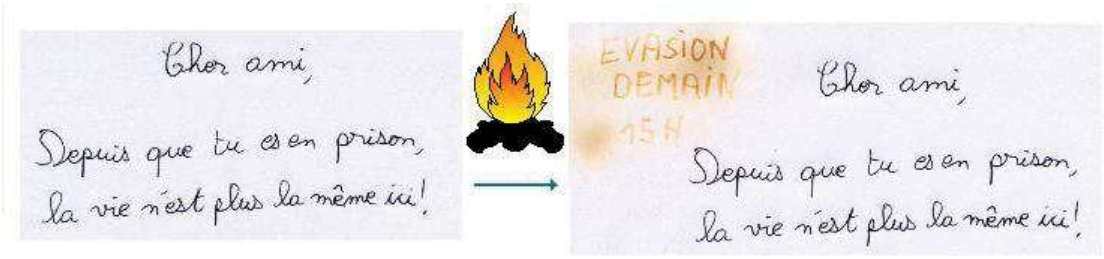



FIGURE 1.2: Examples of ancient steganography using invisible ink.

Apparently neutral's protest is thoroughly discounted and ignored. Ismam hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetable oils.

It seems quite innocuous. Now, by taking the second letter of each word, we get: 'pershing sails from ny June i' (the Pershing leaves New York on June 1). This explains why Americans, who feared very much the use of steganography, censored many communications, including requests to broadcast records on the radio.

## 1.2 Modern steganography

In a world where digital media overtake physical media, ancient steganography is no more, as it has been replaced by modern steganography. We give some possible applications (malicious and legitimate ones) of modern steganography with the following examples.

### Malicious uses

The Internet is an inexhaustible source of digital data, the countless electronic files that circulate on the web and the many digital media that are exchanged via point-to-point communications make it difficult to detect hidden messages in these media.

This may be what makes steganography, despite its multiple applications, always associated with malicious uses. The media report that terrorist organizations

with mafia groups and hackers are the main users of steganography. They use this technology mainly to communicate harmless and dangerous content. We mention here a few examples:

- Steganography for terrorist purposes:

  In February 2001, Wired News mentioned in its article "Secret Messages Come in.Wavs"[1] that steganography technique is widely used over the internet, where they reported the existence of hidden data within images of the eBay and Amazon sites. Terrorists reportedly used steganography to communicate, and Al-Qaida even used it in its preparations for the September 11 attack.

  In 2002, University of Michigan researcher had automated a search for images with hidden content on the Internet. Through their published article, they reveal that out of the 2 million images uploaded to the USENET forums and the eBay auction site, 1% of all images conceal hidden content (about 20,000 images) [Provos and Honeyman, 2002].

- Espionage: People who possess illegal data, such as those related to espionage, want to hide them. Steganography offers them the ideal tool to hide these data in another "innocent" file, a sound, an image, and then save them on their own hard drive in a safe way.

- Computer hacking: The magazine of computer security MISC evokes in its September issue (MISC n9, sept-oct 2003, p.11), malicious use of steganography by hackers: "a hacker can for example code a multi-threaded program and distributed it in memory". The multi-threaded program can be fragmented and sent to the victim hidden in dispatched files using *batch steganography*[2] processes in images. The "reassembly" of the multi-thread program being done locally, just by calling the appropriate place. A hacker may also attempt to hide a Trojan horse or other malicious code in the autorun of a removable media.

[1] The article can found Here

[2] Batch steganography consists of embedding a secret message by spreading it over several innocent files.

### Legitimate uses

In some non-democratic countries where freedom of expression is totally prohibited, steganography appears as a means of communicating more freely. In these countries, the use of steganography is illegal, but its use, in this case, is legitimate.

This art is at measured risk as it applies to information. Despite the fact that these techniques strive at all cost to hide the presence of a secret message, yet they may be detected and compromised by an eavesdropper during the transmission. This eavesdropper will analyze all messages that go through the communication channel with the objective of preventing, modifying or destroying the secret message. The field of research that explores techniques that counter steganography is known as steganalysis.

## 1.3 The prisoners' problem

The classic model of secret or invisible communication was first proposed by Simmons with the "prisoner problem" (Figure. 1.3) [Simmons, 1984].

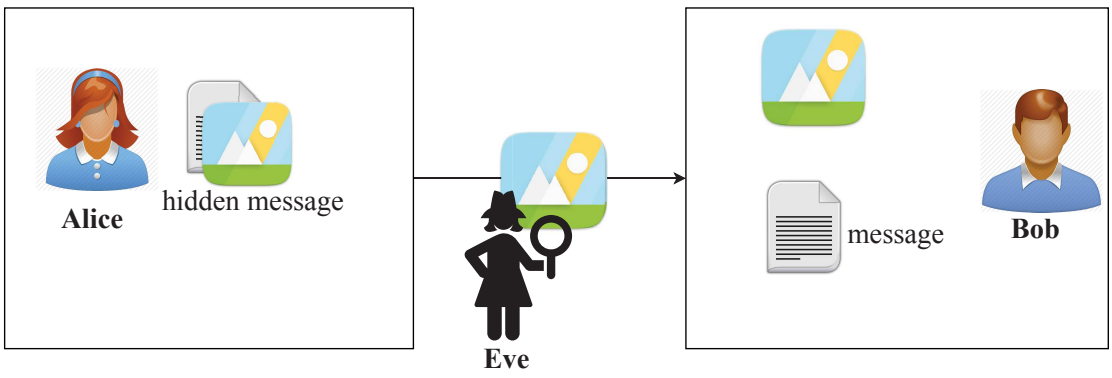



FIGURE 1.3: The general model of Simmons' "prisoner problem".

Alice and Bob are arrested for a criminal offence and placed in two separate cells. They wish to establish an escape plan, but unfortunately, all communication between them is monitored by a warden named Eve. The use of encryption is impossible since any suspicion of secret information exchange would result in an immediate communication cut off and they would be placed in solitary, a confinement where no exchange is possible. Under such circumstances, Alice and Bob need to establish a secret communication in order to communicate without attracting Eve's attention. They can hide the secret information within an innocent message. Eve, in this case, would let the message get through. For instance, Bob could draw a white bird flying in clear sky and send this artwork to Alice. Warden Eve does not know that the colours and shapes in the image mean anything.

Unlike encryption techniques that aim to hide the content of a message, steganography goes further by trying to hide the very fact of communication. Thus, an eavesdropper cannot suspect the existence of secret communication. This makes Simmons' model a perfect choice to illustrate steganography and steganalysis. Alice and Bob represent the two parts of communication that want to exchange information secretly; their task is then steganography. Eve the attentive warden, she is likely to check each message goes through the communication channel, and the mere suspicion of a secret communication would lead to immediate disconnection of that channel, i.e. the failure of steganography. Warden Eve's job is steganalysis. In the literature, three types of steganalysis can be found. These types are identified according to the action that the warden Eve can take when she detects a secret communication. We have steganalysis with the passive warden, steganalysis with the active warden or steganalysis with the malicious warden (explained in Chapter 5).

## 1.4 Notation

In order to make the reading of this manuscript more fluid and to avoid any ambiguity for the reader, we provide this section in which we give notations used

throughout the dissertation. Nevertheless, we do not describe all notations, but rather the notation necessary to proceed. We introduce additional notation if needed.

Scalar $a$: lowercase letter in italic represent scalars.

Set $\mathcal{A}$: calligraphic font is reserved for sets.

Matrix or a vector $\boldsymbol{a}$: Boldface for lower-upper case letter is used for vectors and matrices.

We will define by $\mathcal{C}$ a set of covers, and $\mathcal{S}$ a set of stegos.

**The used Symbols:**

$\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s)}, y^{(s)})\}$: a dataset.

$D(\boldsymbol{x}, \boldsymbol{y})$: the distortion between the cover image $\boldsymbol{x}$ and the stego image $\boldsymbol{y}$.

$f(.)$: a function to return the features vector.

$\boldsymbol{i} = \{(y_1, ..., y_n) \in \{0, ..., 255\}^n\} \in \mathcal{S}$: an image with unknown label.

$\boldsymbol{m} = (m_1, ..., m_m) \in \{0, 1\}^m$: the secret $m$-bit message.

$\boldsymbol{n} = \{n_i \in \{-1, 0, 1\}\}_i^n$: the modification map.

$\boldsymbol{p} = \{p_i \in \mathbb{R}\}_i^n$: a probability map associated with the cover image ($\boldsymbol{x}$).

$P_E$: the probability of error. This notation will be used to compare the performance of the different steganalysis models.

$\boldsymbol{x} = \{(x_1, ..., x_n) \in \{0, ..., 255\}^n\} \in \mathcal{C}$: a grayscale cover image composed of $n = n_1 \times n_2$ pixels.

$\boldsymbol{x}^{(i)}$: an element of the database $\mathcal{D}$

$\boldsymbol{y} = \{(y_1, ..., y_n) \in \{0, ..., 255\}^n\} \in \mathcal{S}$: a stego image composed of $n = n_1 \times n_2$ pixels.

$\boldsymbol{y}_{\boldsymbol{x}\sim\boldsymbol{x}_i}$: the stego image obtained after modification of the $ith$ element $x_i$ of the cover image $x$.

$y^{(i)}$: a label of the element $\boldsymbol{x}^{(i)}$ in the dataset $\mathcal{D}$

$\boldsymbol{z} = \{\boldsymbol{z}_j^{(i)}\}$: a feature vector.

$\boldsymbol{\mathfrak{z}}$: a random noise vector.

$\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$: neural network's parameters. $\boldsymbol{w}$: the vector of weights. $\boldsymbol{b}$: the vector of bias.

$\boldsymbol{\rho} = \{\rho_i \in [0, \infty[\}_i^n$: a cost map associated with the cover image $\boldsymbol{x}$.

# Part I

# State of the art

# Chapter 2

# Machine learning approaches

## Contents

Artificial intelligence (AI) is a research field that groups all techniques that aim to reproduce, through artificial systems, the various human cognitive capacities and more broadly the "intelligent" behaviours of living beings, in particular with regard to their ability to solve complex problems. AI techniques may cover tasks as varied as learning a simple XOR function, recognizing/segmenting objects, ending with self-driving car.

In the past, building an intelligent system consisted in coding a complex program such as searching a tree (to play chess), or performing logical deductions from rules written by experts (medical diagnosis based on symptoms). This "hand-crafted" approach has its limits. It is difficult to apply it to tasks with high complexity level, such as images recognition, or voice recognition. It is virtually impossible to write a robust program that can handle all variables of such complex tasks. This is where machine learning is required.

Machine learning is a sub-domain of AI that aims at providing intelligent systems with the ability to self-learn, from data and examples, solving complex problems. The principle is not to use "hand-coded" programs, but a well-designed model that can analyze data in order to find distinctive patterns for a specific task.

The classical machine learning systems are generally composed of two blocks: a feature extractor, followed by a trainable classifier. The feature extractor is a program designed by an expert in the field of application. It is used to extract discriminating features from the raw data. The classifier is another program which is used to learn, from the extracted features, a function most suitable for the desired objective (in classification tasks, the objective is to define the best separation between the different classes).

Although these systems are astonishingly effective in addressing a wide variety of complex problems, they do have their limitations. Building a well-adapted feature extractor is very difficult, as it requires the knowledge and expertise of a qualified expert in the target field. Moreover, the extracted features cannot be extended from one problem to another. One way to overcome these problems is through "deep learning". As illustrated in Figure. 2.1, deep learning is a subclass of machine

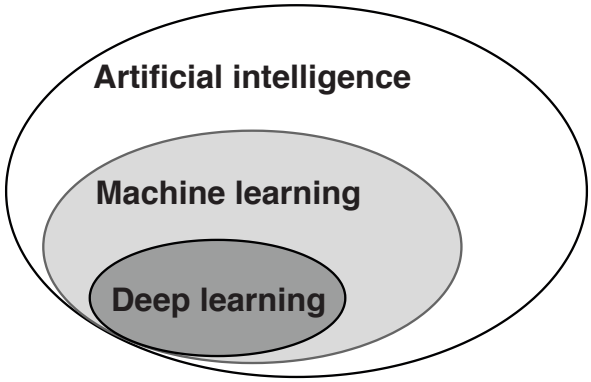



FIGURE 2.1: The global frame of artificial intelligence.

learning methods that combines the two-learning blocks of a traditional machine learning approach (feature extractor, classifier) into one block represented by a neural network, that "automatically" learns the best extractor for a given task. Machine learning and deep learning share the same concept of learning complex tasks from a given data set. Depending on the nature of this data set, two main learning classes can be identified, the *supervised learning* and *unsupervised learning*.

### Supervised learning

Let $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s)}, y^{(s)})\}$ be a given dataset where $s$ is the number of samples in the database, $\boldsymbol{x}^{(i)}$ is a data sample, and $y^{(i)}$ its associated label. The aim of supervised learning is to learn a function that best approximates the relationship between an input $\boldsymbol{x}^{(i)}$ and its label $y^{(i)}$. As an example of supervised learning approach, we can mention supervised classification, regression.

### Unsupervised learning

Unlike supervised learning, for non-supervised learning no annotated data is required and thus the dataset is limited to $\mathcal{D} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(s)}\}$. This type of learning approach strives to find structures, dependencies between the given samples; in other words, unsupervised learning task is to find the most appropriate representation of the data. The most well known non-supervised approach is the K-mean.

In this thesis, we focused in particular on supervised learning, more specifically on supervised classification; thus, all techniques and methods presented in this chapter are oriented towards classification purposes. In the next sections, we describe more accurately the two-step machine learning approaches, as well as the one-step deep learning approaches.

## 2.1 Machine learning: Two-step learning approach

The term machine learning was first introduced in 1959 by Arthur Samuel [Samuel, 1959]. It refers to a specific field of computer science that *provides computers with the ability to learn without being explicitly programmed.*

Until recently, traditional machine learning systems were made of two steps: a feature extractor, followed by a simple classifier trained using a data set $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s)}, y^{(s)})\}$ (see Figure. 2.2).

**Feature extractor**

In computer vision, features extraction consists of a mathematical transformation of a data sample $\boldsymbol{x}^{(i)}$ (digital image, sound, ...etc.) into an $m$-dimension feature vector $\boldsymbol{z}^{(i)} \in \mathbb{R}^m$ with $i \in \{1, s\}$. Each component of $\boldsymbol{z}^{(i)}$ corresponds to a describing feature of $\boldsymbol{x}^{(i)}$. The main goals of features is first to reflect the discriminating properties of $\boldsymbol{x}^{(i)}$ relatively to the classification problem, and second to reduce $\boldsymbol{x}^{(i)}$ dimensionality. Formally, the features vector $\boldsymbol{z}^{(i)}$ can be written as follow:

$$\boldsymbol{z}^{(i)} = f_{ext}(\boldsymbol{x}^{(i)}), \tag{2.1}$$

where $f_{ext}(.)$ is a transformation function applied by the feature extractor.

**Classifier**

With the help of a feature extractor, a dataset $\mathcal{D} = \{(\boldsymbol{z}^{(1)}, y^{(1)}), ..., (\boldsymbol{z}^{(s)}, y^{(s)})\}$ is created, where $\boldsymbol{z}^{(i)} = \{\boldsymbol{z}_j^{(i)}\}$ is a feature vector with $i \in \{1, s\}$ and $j \in \{1, m\}$. This dataset is sent to a trainable classifier (illustrated in Figure. 2.2.) in order to learn

a mapping function $f_{classif}(.)$. This function takes a feature vector $\boldsymbol{z}^{(i)}$ as input and outputs a discrete value (prediction) $\hat{y}^{(i)} \in \{0, c-1\}$, where $c$ is the number of classes of the problem. Classifier training consists in making the predicted value, $\hat{y}^{(i)}$, for a given feature vector $\boldsymbol{z}^{(i)}$, equal to the desired label $y^{(i)}$. The trained classifier is then used for a prediction phase where no training is needed.

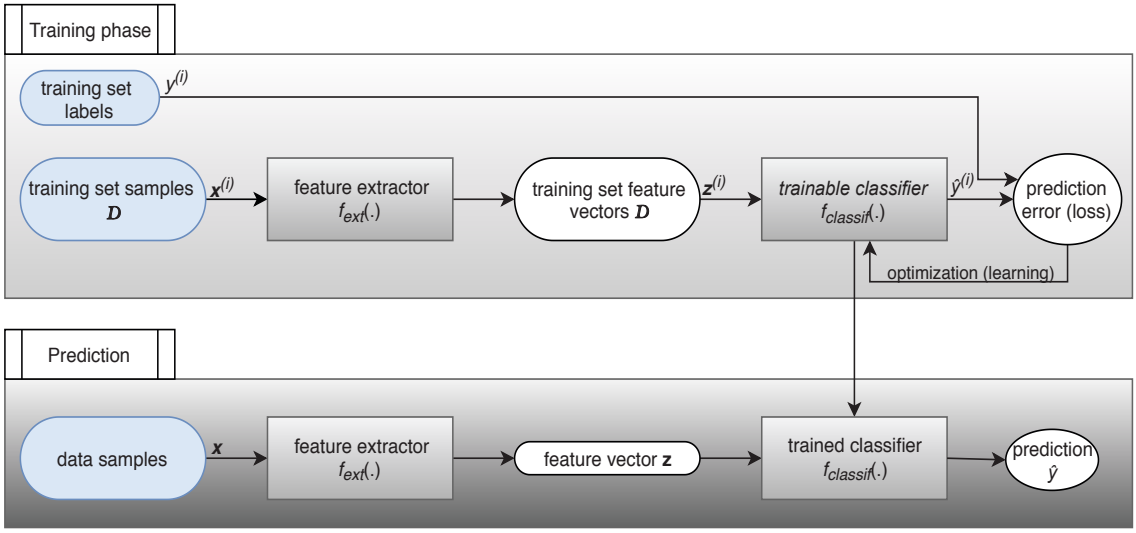



FIGURE 2.2: The two-step learning approach for a classification problem.

$$\hat{y}^{(i)} = f_{classif}(\boldsymbol{z}^{(\boldsymbol{i})}) = \sum_{j}^{m} z_j^{(i)} \times w_j + b, \tag{2.2}$$

A common type of classifier is logistic regression. It operates, as shown in Equation. 2.2, by calculating a weighted sum with the component of the features vector. Each component is multiplied by a weight (positive or negative) before being summed. Depending on the comparison of this sum with a threshold, the class is decided. The weights $\boldsymbol{w} \in \mathbb{R}^m$ form a kind of "prototype" for the class to which the feature vector is correlated. The weights are different for each category, and they are the parameters updated during the training phase.

The two-step learning approach for a classification problem can be summarized as illustrated in Figure. 2.2. First, we design a feature extractor that extracts the relevant features from the raw data. Secondly, we form a classifier using labelled data (features and their corresponding labels). Finally, after the training phase, we obtain a trained model that can be directly used for prediction.

## 2.2 Deep learning: One-step learning approach

In this section, we briefly present the development of an artificial Neural network (ANN) by introducing the main concepts, models and relevant algorithms. For a more detailed review of the field, we recommend reading [Goodfellow et al., 2016].

### 2.2.1 Artificial Neuron

In 1943, McCulloch and walter Pitts had proposed a computational model inspired of the biological neuron [McCulloch and Pitts, 1943]. In the present time, this model is known as **artificial neuron**. The progression from biological to artificial neurons can be summarized in Figure. 2.3. In a biological neuron, the **dendrites** receive signals, the frequency of each signal depends on the **synapse**. These signals are then transmitted to the body of the cell, which is characterized by an activation condition. It is by checking the activation condition that the **axon** is activated to emit signals again to the neurons that follow through their dendrites. In the artificial neuron, dendrites are represented by input values, the synapse by the weight of each input, the body cell is represented by the activation function and the axon by the output.

An artificial neuron is a mathematical unit that maps data inputs to an output value which, by its turns, may be passed as an input to another neuron. With the help of a vector of weights $\boldsymbol{w} = (w_1, ..., w_m)$ where $w_i \in \mathbb{R}$, artificial neurons process the input vector $\boldsymbol{x} = (x_1, ..., x_m)$ as in Equation. 2.3 and then applies a non-linear function $f(.)$ (known as activation function) to produce a scalar output $\hat{y}$ according to the following function:

$$\begin{aligned} o &= \sum_{i=1}^{n} w_i \times x_i + b \\ \hat{y} &= f(o) \end{aligned} \tag{2.3}$$

where $o$ is the output of the weighted sum of the input vector $\boldsymbol{x}$ plus a bias $b$.

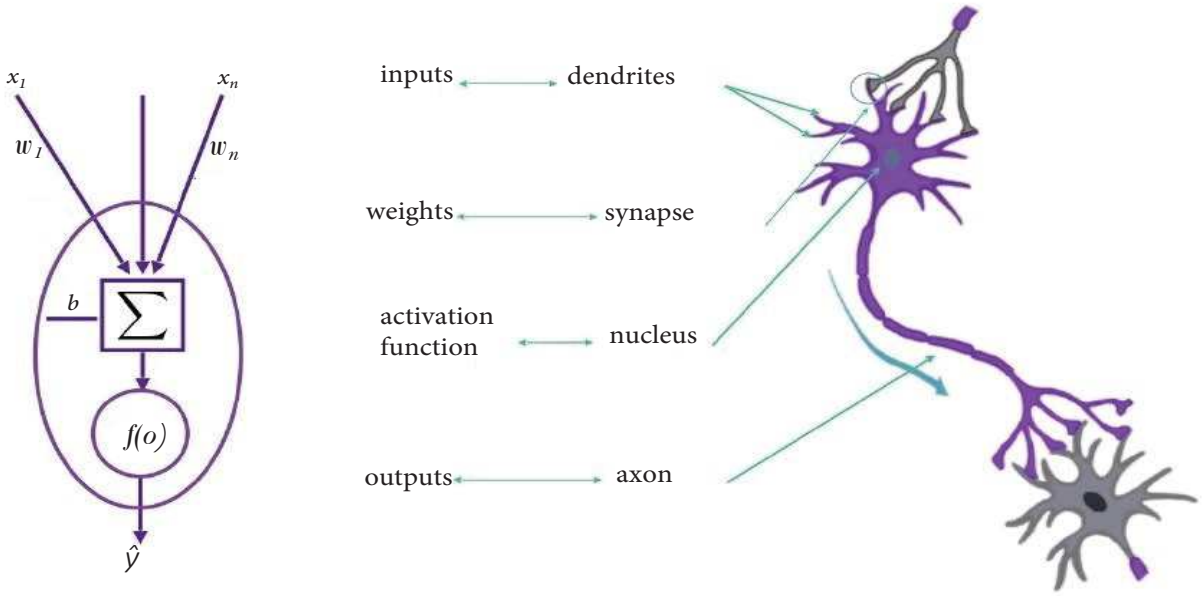



FIGURE 2.3: Comparison between a biological neuron and an artificial neuron. Extracted from [Ghannay, 2017].

The most famous artificial neuron example is the well-known perceptron.

**The Perceptron** Introduced in [Rosenblatt, 1958], the perceptron is the first artificial neuron that was used for pattern recognition. Its first implementation used hardware transistors (therefore not derivable) to compute the weighted sum of the inputs.

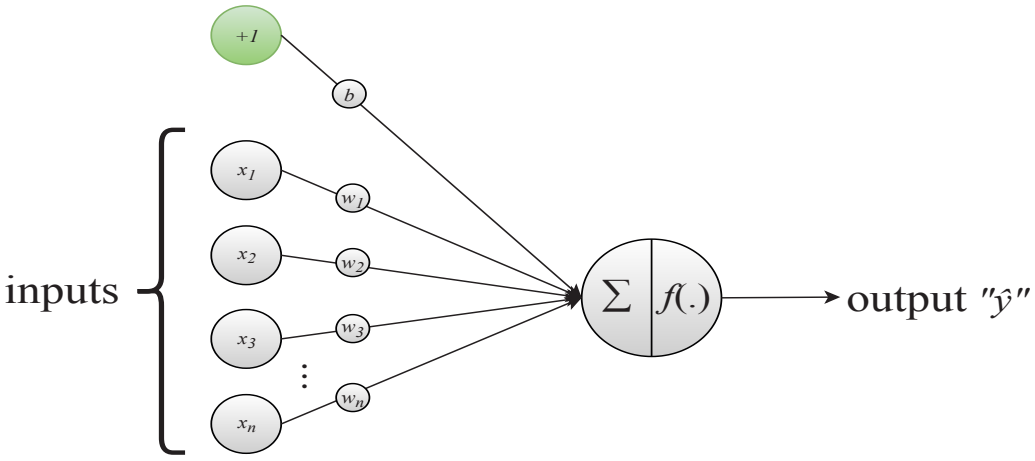



FIGURE 2.4: Perceptron model.

In Figure. 2.4 we give an example of perceptron, that takes as inputs a vector $x = (x_1, ..., x_m)$ and outputs a prediction $\hat{y}^{(j)}$. Each element of $x^{(j)}$ is associated

with a a weight $w_j$ that modulates its importance. Same as an artificial neuron, a perceptron's objective is to learn the weights vector $\boldsymbol{w} = (w_1, ...w_m)$ such that it best approximates a mapping $F : X \to Y$. To this end, after a random initialization of the weight vector $\boldsymbol{w}$, the perceptron, proceeds by repeating the following steps across the entire database $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s)}, y^{(s)})\}$.

1. Select a training example $\boldsymbol{x}^{(i)} = (x_1, ..., x_m) \in \mathcal{D}$, and its ground truth binary label $y^{(i)} \in \{0, 1\}$,

2. compute the perceptron's prediction, with regard to the input $\boldsymbol{x}^{(i)}$, using the following equation:

$$\hat{y}^{(i)} = f(\sum_{j=1}^{m} x_j^{(i)} \times w_j + b), \tag{2.4}$$

   where $f(.)$ is the activation function. Among the most used activation function, we find the Heaviside function (known as the unit step function):

$$Heaviside\,(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{2.5}$$

3. update all perceptron weights $\boldsymbol{w}$ over time with respect to the difference between the ground truth label $y^{(i)}$ and the predicted one $\hat{y}^{(i)}$:

$$\begin{aligned} w_j(t+1) &= w_j(t) + \Delta w_j, \\ \Delta w_j &= \lambda(y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}, \end{aligned} \tag{2.6}$$

with $\Delta w_j$ the amount of changing of the weight value, and $\lambda$ the learning rate (a value greater than 0 controlling how much the weights change), $t$ is the iteration step whit $t = 0$ is the initialization step.

When it was created, the perceptron appeared to be a major step forward in the development of artificial intelligence. Unfortunately, the technical limits were

quickly reached. Authors in [Minsky and Papert, 1969] have demonstrated that XOR function can not be expressed with a perceptron, because a perceptron works only on linearly separable data as shown in Figure. 2.5 (b). Indeed, a single straight line is enough to divide the inputs into their correct categories. However,

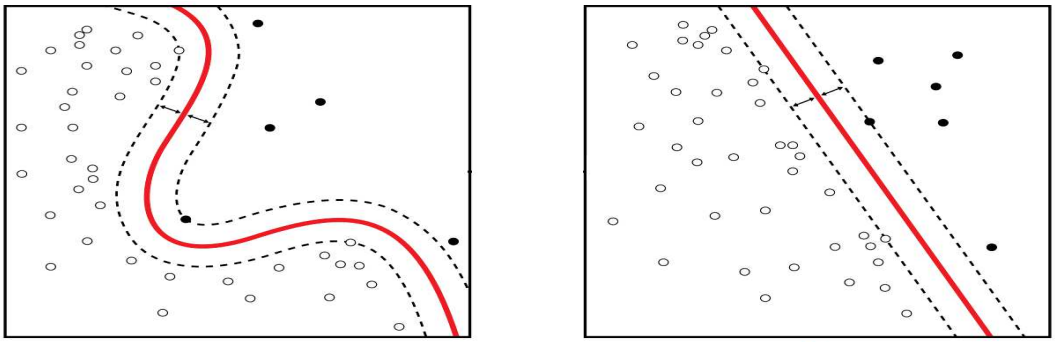



FIGURE 2.5: Perceptron model.

it was later discovered that the use of a more complex model consisting in many perceptrons would allow circumventing the stated limitation of classification of non-linearly separable data (where splitting the inputs into their correct categories using a straight line or a plane is impossible (Figure. 2.5 (a))). This complex model is known as **artificial neural network**.

### 2.2.2 Artificial Neural Networks (ANN)

Artificial Neural Networks are a widely used method today. Thanks to their success and their promising results in many different fields, for both supervised and unsupervised learning, ANN has been identified as the dominant algorithms of machine learning.

Despite the fact that ANN goes back to 1943 [McCulloch and Pitts, 1943], they get to become famous a bit later. This is thanks to the arrival of new architectures such those in [Hopfield, 1982, 1984], and new learning methods like back-propagation algorithm [Rumelhart et al., 1986, 1988, Lecun, 1988] (to be discussed later in Section. 2.2.5) that allows ANN to learn efficiently. However, the true potential of

ANN has revealed later through the provision of powerful computing tools (such as graphics processors).

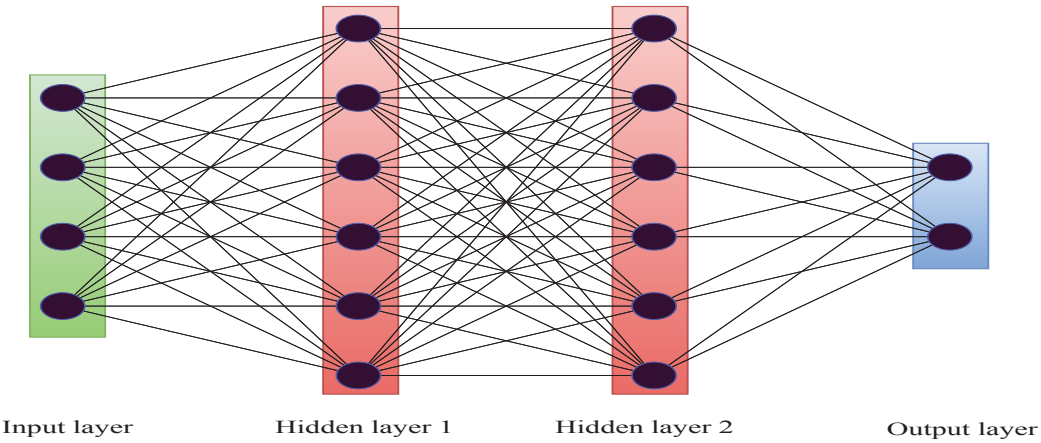



FIGURE 2.6: Artificial neural network.

Artificial Neural Networks, are made up of a succession of $L$ layers, each layer $l \in \{0, ..., L\}$ is composed of $N_l$ neurons (presented in Section. 2.2.1). Generally, as illustrated in Figure. 2.6, layers are related to each other by weighted connections. For instance the neuron $n_j^{[l]}$ from the $l$-th layer is connected, through a weighted connection $w_{j,k}^{[l],[l+1]}$, to the neuron $n_k^{[l+1]}$ from $(l+1)$-th layers, where $j \in \{1, ..., N_l\}$, $k \in \{1, ..., N_{l+1}\}$ are neuron indexes in a layer and $l$, $(l+1)$ are layers indexes. All these weights $w_{j,k}^{[l],[l+1]}$ form the weight vector $\boldsymbol{w}$.

In general, an ANN is composed of three types of layer.

1. Input layer (0-th layer): The neurons in the input layer (green layer in Figure. 2.6) are fed with the input data that is supposed to describe the problem to be analyzed. Taking the example of steganalysis, these neurons will be associated with the image's pixels.

2. Hidden layers: These are all layers between the input layer and the output layer, and illustrated with red layers in Figure. 2.6. The hidden layers roles are to perform calculations and transmit the information from the input layer to the output layer. The choice of the number of hidden layers within an ANN is crucial, but it is not implicit and must be adjusted. In general,

the complexity of a neural network model is defined by the number of its hidden layer (know as network depth).

3. Output layer: It is the last ($L$-th) layer of the network (blue layer in Figure. 2.6). This layer contains the artificial neurons that are associated with the model outputs. Each neuron corresponds to one target class. The number of neurons in this layer is equal to the number of classes ($N_L = c$). In the example of steganalysis, this layer will contain two neurons, one for class **cover** and the other for class **stego**.

### 2.2.3 ANN different architectures

There are many types of artificial neural networks (ANN), in what follows, we will discuss the architectures of some of the most known types.

**Multi-layer Perceptrons** A world-wide known example of an artificial neural network is a multilayer perceptron (MLP), an oriented connection graph (i.e. architecture) of a succession of layers of artificial neurons [Rumelhart et al., 1988]. Every neuron of the $l$-th layer is connected to all neuron of the $(l+1)$ layer. This explains why MLP is also referred to as a fully connected network.

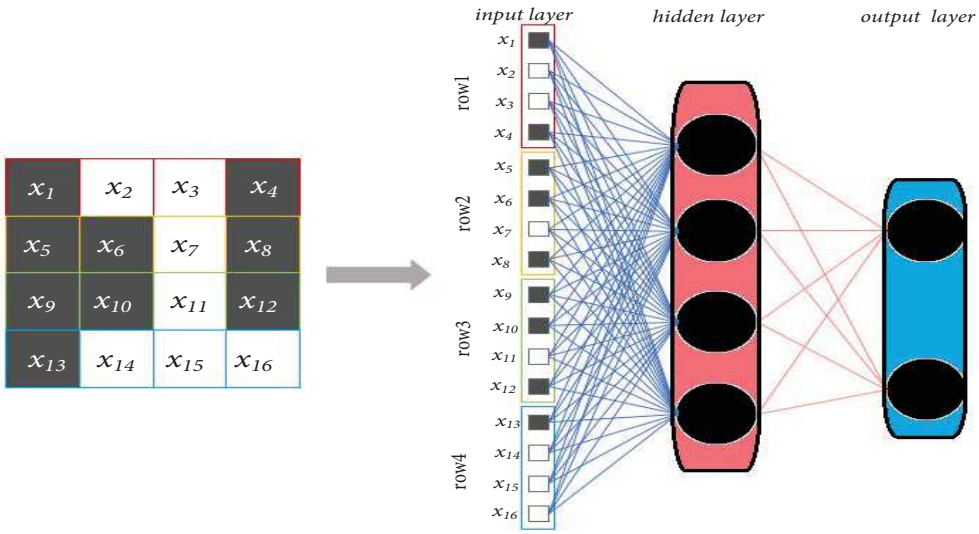



FIGURE 2.7: Multilayer perceptron.

When a data sample $\boldsymbol{x} = (x_1, ..., x_m)$ is fed to the input (0-th) layer of an MLP (each neuron $n_k^{[0]}, k \in \{1, ..., N_0\}$ is associated with all elements of the flattened input $x_j$ as shown in Figure. 2.7). The model afterwards computes the output of each neuron in all layers one by one until the last ($L$-th) layer. Considering a neuron $n_k^{[l+1]}$, the $k$-th neuron of the $(l+1)$-th layer, its output is calculated as following:

$$n_k^{[l+1]} = f(\sum_{j=1}^{N_l} n_j^{[l]} \times w_{j,k}^{[l],[l+1]} + b_k^{[l+1]}), \tag{2.7}$$

with $f(.)$ a non-linear function (will be discussed later in Section. 2.2.6.2, while $w_{j,k}^{[l],[l+1]}$ is the weighted connection between the neurons $n_j^{[l]}$ and $n_k^{[l+1]}$, $N_l$ is the number of the neurons in the $l$-th layer.

The training of an MLP consists in adjusting its parameters $\boldsymbol{\theta}$=($\boldsymbol{w}, \boldsymbol{b}$) for a given problem (i.e. a multi-class classification problem) with respect to the training data. The weight adjustment can be considered as an optimization problem. Therefore, several optimization algorithms can be used for the training of an MLP. Authors in [Werbos, 1974, Lecun, 1988, Rumelhart et al., 1986] have shown that MLP can be effectively trained using stochastic gradient descent through **back-propagation**, which will be further discussed in Section. 2.2.5.

**Auto-encoder** (AE) is another example of artificial neural network. It is first introduced in the 1980s by Hinton in [Rumelhart et al., 1986] but they gained attention later with [Hinton et al., 2006, Bengio, 2009]. In Figure. 2.8, we give the architecture of a simple AE composed of three layers ($L$=3). The first layer, with the hidden layer forms the encoder part, while the hidden layers, and the last layer, form the decoder part.

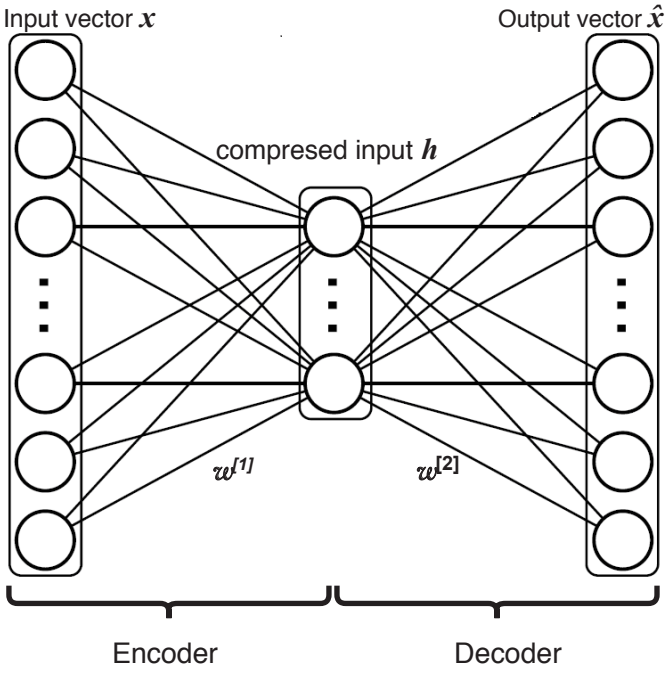



FIGURE 2.8: An auto-encoder composed of an input layer, a hidden layer and an output layer. For reasons of readability, biases are omitted.

From an input sample $\boldsymbol{x}$, the encoder calculates an $N_h$-dimension encoded vector $\boldsymbol{h}$ (with $N_h$ the number of neurons in the hidden layer) as follows:

$$h_j = f(\sum_{k=1}^{m} w_{k,j}^{[1]} \times x_k + b_j^{[1]}), \tag{2.8}$$

where $\boldsymbol{w}^{[1]}$ is a matrix of size $N_h \times m$ and $\boldsymbol{b}^{[1]}$ a bias vector of size $N_h$. $f(.)$ is an activation function.

The decoder tries to reconstruct vector $\boldsymbol{x}$ from the encoded vector $\boldsymbol{h}$. The resulting reconstructed vector is noted as $\hat{\boldsymbol{x}}$ and computed as follow:

$$\hat{x}_j = f(\sum_{k=1}^{N_h} w_{k,j}^{[2]} \times h_k + b_j^{[2]}), \tag{2.9}$$

where $\boldsymbol{w}^{[2]}$ is a weight matrix of size $m \times N_h$ and $\boldsymbol{b}^{[2]}$ is an $m$-element bias vector.

The objective of an auto-encoder consists in reducing a reconstruction error between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$. To this end, AE minimizes the mean square error $MSE(\boldsymbol{x}, \hat{\boldsymbol{x}})$ by adjusting the parameters with the set of parameters $\boldsymbol{\theta} = \{\boldsymbol{w^{[2]}}, \boldsymbol{b^{[2]}}, \boldsymbol{w^{[1]}}, \boldsymbol{b^{[1]}}\}$, where

$$MSE(\boldsymbol{x}, \hat{\boldsymbol{x}}) = ||\boldsymbol{x} - \hat{\boldsymbol{x}}|| \tag{2.10}$$

Another version of AE is the stacked auto-encoder (SAE) presented in [Vincent et al., 2010]. It is like a simple autoencoder but with more hidden layers. The encoded vector $\boldsymbol{h^{[l]}}$ of the $l$-th layer is calculated using the outputs of $(l-1)$-th layer as follows:

$$h_j^{[l]} = f(\sum_{k=1}^{N_{l-1}} w_{k,j}^{[l-1],[l]} \times x_k^{[l-1]} + b_j^{[l]}), \tag{2.11}$$

with $\boldsymbol{h^{[0]}} = \boldsymbol{x}$ the input vector. To train an SAE we use a "*Greedy Layer-Wise Pretraining*", where each layer $l$ is pre-trained as a basic auto-encoder. Then the learned vector $\boldsymbol{h^{[l]}}$ is retrained and used to train the next layer $l+1$ and so on until the desired level of abstraction is reached.

ANNs like those presented above are called shallow networks (with a limited number of layers). These networks are powerful learning models. However, to represent more complex features and "learn" increasingly sophisticated models, more complex ANN architectures are required. This can be effectively achieved by simply increasing the number of hidden layers and thus introducing a depth notion to ANN (such as SAE compared to a simple AE). These deep ANNs are studied in a field called **deep learning**.

### 2.2.4 Deep learning

Deep neural network, as presented in Figure. 2.9.A, is nothing more than a deeper version of a simple ANN, shown in Figure. 2.9.B. The depth notion is related to

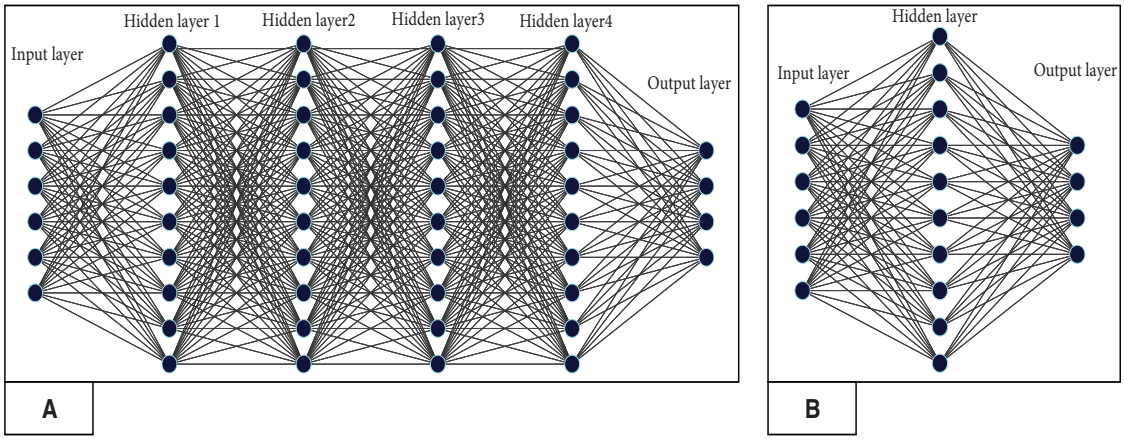



FIGURE 2.9: a side-by-side comparison between a deep neural network (A) and a shallow neural network (B).

the number of hidden layers. The more layers a network has, the more features it can learn. These features are then combined in the output layer in order to make a prediction. The role of hidden layers is then to transform the input data into a more abstract representation.

Since the deeper the network is, the more neurons there are and therefore, the more weights to learn, deep neural networks require massive datasets and a considerable training time compared to an ANN. Here and below, we will be referring to the deep neural network as "artificial neural network" (ANN).

### 2.2.5 Artificial neural networks training

The training (or training phase) of a shallow or deep neural network consists in adjusting its parameters $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$, where $\boldsymbol{w}$ represents the weight matrix and $\boldsymbol{b}$ the bias vector, while trying to correctly classify the samples of a given learning set $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s)}, y^{(s)})\}$, and their associated labels. Computing the set of parameters $\boldsymbol{\theta}$ can be modelled as an optimization problem.

One of the most widely used algorithm for ANN training is the **backpropagation**. It was proposed for the first time in [Werbos, 1974] but adopted for artificial neural network training later in [Rumelhart et al., 1986, Lecun, 1988]. It is based on the minimization of a derivable function called the loss function, $\mathcal{E}$, using the

gradient descent method. The loss function is calculated between $\hat{y}^{(i)}$, the ANN's prediction given an input sample $\boldsymbol{x}^{(i)}$, and the ground truth label $y^{(i)}$. Many types of loss functions can be considered, such as root mean square error or cross-entropy [Stemmer et al., 2002].

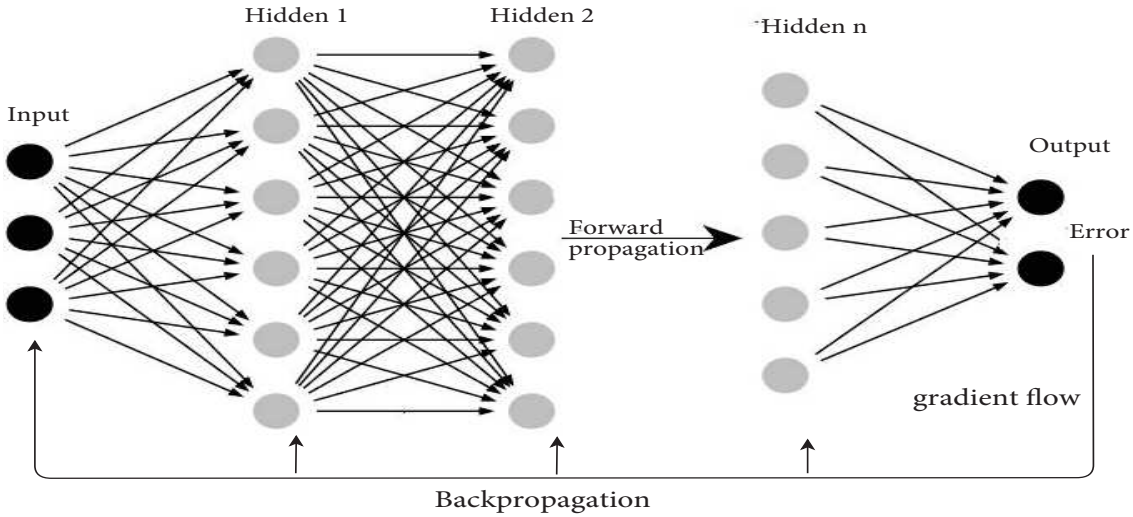



FIGURE 2.10: ANN training protocol.

As illustrated in Figure. 2.10, the back-propagation is preceded by a forward propagation during which the ANN will use the the current weight matrix, $\boldsymbol{w(t)}$, and bias vector $\boldsymbol{b(t)}$ vector to make a prediction $\hat{y}^{(i)}$ for a given input $\boldsymbol{x}^{(i)}$. The error between prediction $\hat{y}^{(i)}$ and the ground truth label $y^{(i)}$ is then computed using a loss function $\mathcal{E}$, such as the $L1$ loss (Mean Absolute Error).

$$\mathcal{E} = MAE(y^{(i)}, \hat{y}^{(i)}) = \sum_i ||\hat{y}^{(i)} - y^{(i)}|| \tag{2.12}$$

Next the back-propagation is carried out by back-propagating the partial derivative of the error, with respect to the parameters $\boldsymbol{\theta} = (\boldsymbol{w}(t), \boldsymbol{b}(t))$, while updating, in the opposite direction of the gradient, $\boldsymbol{w}(t)$ and $\boldsymbol{b}(t)$, with the help of gradient

descent algorithms, as follow:

$$\begin{aligned}
&\frac{\partial \mathcal{E}}{\partial \boldsymbol{w}(t)} = \Delta \boldsymbol{w}, \\
&\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \lambda \Delta \boldsymbol{w}, \\
&\frac{\partial \mathcal{E}}{\partial \boldsymbol{b}(t)} = \Delta \boldsymbol{b}, \\
&\boldsymbol{b}(t+1) = \boldsymbol{b}(t) - \lambda \Delta \boldsymbol{b},
\end{aligned} \tag{2.13}$$

with $\lambda$ the learning rate.

The partial derivative $\frac{\partial \mathcal{E}}{\partial \boldsymbol{w}}, \frac{\partial \mathcal{E}}{\partial \boldsymbol{b}}$ in Formula. 2.13 are calculated consecutively, layer by layer, from the output layer to the input layer following the chain rule [1]. The updates of the parameters $\boldsymbol{\theta}$ (cf. Equation. 2.13 ) are computed either based on each training sample (fully stochastic training) or based on small batches of several training examples (mini-batch training).

For $t = 0$ the weights $\boldsymbol{w(t = 0)}$ are randomly initialized according to a given distribution. The most commonly used types of distributions are the Gaussian, the uniform or Xavier's distribution [Glorot and Bengio, 2010].

The learning rate $\lambda$ determines the updating step of the ANN parameters (weight and bias). Its values must be chosen wisely since a low learning rate offers training stability by reducing the chances of missing a good minima, but in that case, training may take a lot of time. On the other hand, a large value of $\lambda$ speeds up the training but can also lead to a learning instability as the loss function fluctuates around the minimum. A rule of thumb is to start training with a relatively high $\lambda$ value. Then, after a sufficient number of iteration, this value is reduced in order to make the update less abrupt. It is important to note that there are some algorithms that change $\lambda$ value in an automatic and clever way, such as Adagrad [Duchi et al., 2011], Adadelta [Zeiler, 2012].

[1] The chain rule is a technique to differentiate composite functions.

To resume, The learning or training consists of applying Equation. 2.13 $T$ times in order to find a good set of parameters $\boldsymbol{\theta(t = T)}$, that delivers the best result. One iteration is a forward/backward passage of a small part of the training data (example/lot/mini-lot). However, the term *epoch* is the most commonly used (rather than iterations) in order to describe the necessary amount of time to train a network. An epoch is composed of many iterations and consists of one forward/backward passage of *all* training dataset.

The number of epochs, the learning rate value, the number of hidden layers, the activation function...etc, are called the *hyper-parameters*. These parameters are decisive for defining a well-designed ANN model. Despite the importance of these parameters, they are usually set manually. To this end, many validations are done, and the set of the hyper-parameters giving the best results are kept. The learned model's generalization is then evaluated on a test set (during the test phase).

**Reflection about the use of ANN for images inputs** With the exception of small object-centred images [Lecun et al., 1995]), neural networks, presented in the previous section, have proven their inability to handle tasks that include images, such as image classification, and object recognition. This is due to their architecture specificity (the neurons from one layer are fully connected to the neurons of the next layer).

For instance, considering an image of size $256 \times 256$, if this image is to be passed to an ANN with only one hidden layer of 512 neurons, the image is then flattened into a vector of $256 \times 256 = 65536$ elements. Each element is assigned to a neuron on the input layer. By linking the input layer to the hidden layer, we obtain more than 33 million weighted connections ($\boldsymbol{w}$ size is $65536 \times 512$). Very quickly, the images become too large for this type of network, the number of parameters explodes, and the network no longer holds in memory. Furthermore, when the image is flattened (to be fed to the input layer), certain key features of the image can be lost, such as the spatial and colour structures. This is a very serious problem because such

features may be necessary for the learning process. To overcome this problem, a new ANN architecture is used, the **Convolutional Neural Network** (CNN).

### 2.2.6 Convolutional Neural Network (CNN)

Convolutional Neural Networks, known as CNN, is a particular form of ANN whose connection architecture is inspired by that of the visual cortex of mammals [Fukushima, 1975]. Although, unlike ANN, CNN has proven effective handling images processing tasks such as classification, detection and segmentation. First CNN used for object detection dates back to 1989 [Lecun, 1989], which was later called LeNet-5 [Lecun et al., 1995, Lecun et al., 1998] (Figure. 2.11). The architectures of current CNNs are quite similar to that of the LeNet-5 but more complex.

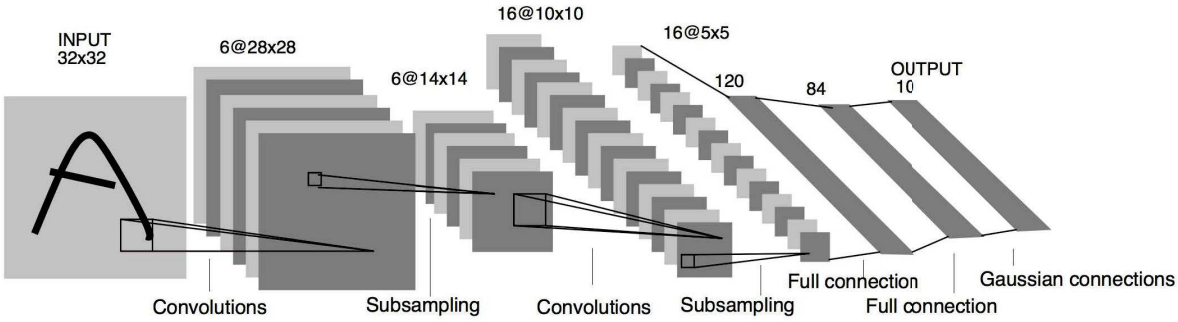



FIGURE 2.11: The architecture of Lenet-5 CNN [Lecun et al., 1998].

As illustrated in Figure. 2.12, a CNN architecture can be split in two parts called *convolutional module*, and *classification module*.. The convolutional module is the one connected to the **input layer** that receives the input image. This module is basically composed of a set of **blocks** which, in their turn, consist in a set of **layers** (usually, but not necessarily, layers are in the following order: convolution layer → activation layer → normalization layer→ pooling layer), with convolution layer being the main building-block. This module works as a feature extractor as it transforms a given image into a feature vector.

This vector is then connected to the input of the **classification module**. It consists in a set of fully connected layers (the MLP of Section. 2.2.3). The role

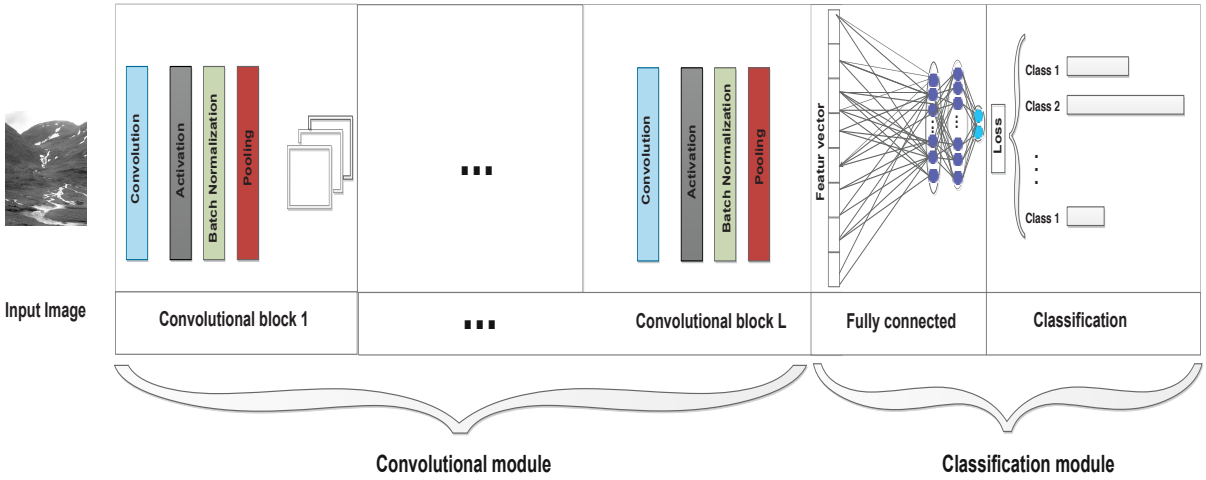



FIGURE 2.12: Two modules of a CNN.

of this classification module is to combine the obtained features to classify the input image. The last layer of this module will provide the CNN output (the prediction). The numerical values of the obtained output are usually normalized, between $[0, 1]$, with the help of a softmax function.

We discuss in the following the basic components of convolutional neural networks, highlighting the different layers used over convolutional and classification modules. It is important to note that in the next section, for better understanding, we will consider that each convolutional block includes only one layer of the same type, which is not necessarily the case in the literature.

#### 2.2.6.1 The Convolution layer

The convolution layer is the key component of convolutional neural networks and the main element of the convolutional block. Its purpose is to identify the presence of a set of features in the input image with the help of a convolution operator.

**Convolution** aims at replacing a pixel value $I_{(i,j)}$ by a weighted sum of its neighbors. The set of weights define what we call a kernel $K \in \mathbb{R}^{(2k_1+1)\times(2k_2+1)}$ where $k_1$ and $k_2$ are the horizontal and vertical size of the neighborhood. This operation produce an output $O_{(i,j)}$ according to the following equation:

$$O_{(i,j)} = (I \star k)_{(i,j)} = \sum_{u=-k_1}^{k_1} \sum_{v=-k_2}^{k_2} I_{(i+u,j+v)} K_{(u,v)} \tag{2.14}$$

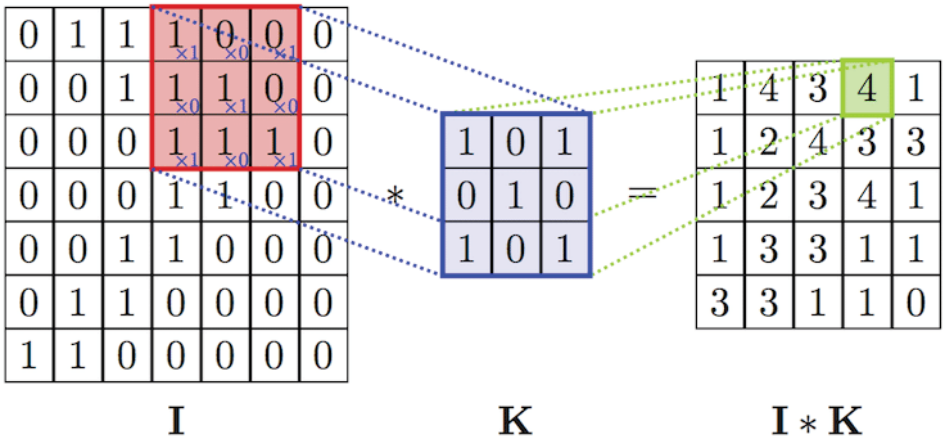



FIGURE 2.13: Convolutional neural network Kernel.

Considering a CNN with $L$-convolutional block, let $l \in \{1, ..., L\}$ be the block index. The convolutional layer of the $l$-th block includes $M^{[l]}$ filters (also called convolution kernels). Each filter $k_m^{[l]}$, with $m = \{1, ..., M^{[l]}\}$, produces a filtered image $F_m^{[l]}$ called *feature-map* [Jarrett et al., 2009] by "sliding" the kernel over the resulting feature-output $F^{[l-1]}$ of the $(l-1)$-th block as illustrated in Figure. 2.13 and according to the Equation. 2.15. However, the output of the $(l-1)$-th convolutional layer may be composed of more than one feature-map in a kind of multi-channel image, in this case, Equation. 2.15 is no more valid, it needs to be rewritten as follow:

$$F_m^{[l]} = F^{[l-1]} \star k_m^{[l]} \tag{2.15}$$

with $k_m^{[l]}$ is a 3D kernel composed of $M^{[l-1]}$ times the same 2D kernel $k_{i,m}^{[l]}$. In the case of $l = 1$, the raw image $I$ is given as the input $F^{[0]} = I$ (see Figure. 2.11).

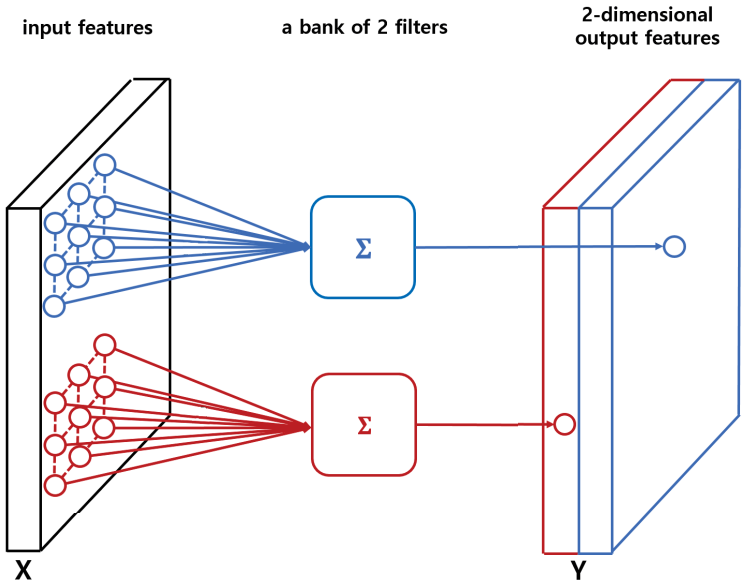



FIGURE 2.14: Convolutional neural network Kernel
(from : https://www.uihere.com/free-cliparts).

If we compare Equation. 2.7 with Equation. 2.15, we can notice that kernels are the trainable parameters of the convolutional layer, i.e. the weights for an ANN. However, when the CNN's use *one* kernel per feature-map, ANN use *several* weights per neuron-output. This explains why CNN's have considerably fewer parameters compared to conventional ANN. Since whatever the neuron, the weights are the same (for the same kernel), the CNN convolution is referenced as **weight sharing**.

#### 2.2.6.2 Activation layer

A non-linear function follows every convolutional layer on each block. It consists in introducing a non-linearity into the network in order to take advantage of the many stacked convolutional layers. Put it differently, without the non-linearity; the CNN would behave like a single-layer perceptron, regardless of the number of convolutional layers it has, since stacking linear function (filtering performed during convolutions) would only give another linear function.

An activation layer is defined by its non-linear activation function. It takes as inputs the generated feature-maps from the convolutional layer $F^{[l]}$ , and outputs

another features-map called *activation-map* $A^{[l]}$. The activation function is an element-wise operation. Therefore, the dimensions of the input features-map and the resulting *activation-map* are identical (see equation 2.16).

$$A^{[l]} = f(F^{[l]}), \tag{2.16}$$

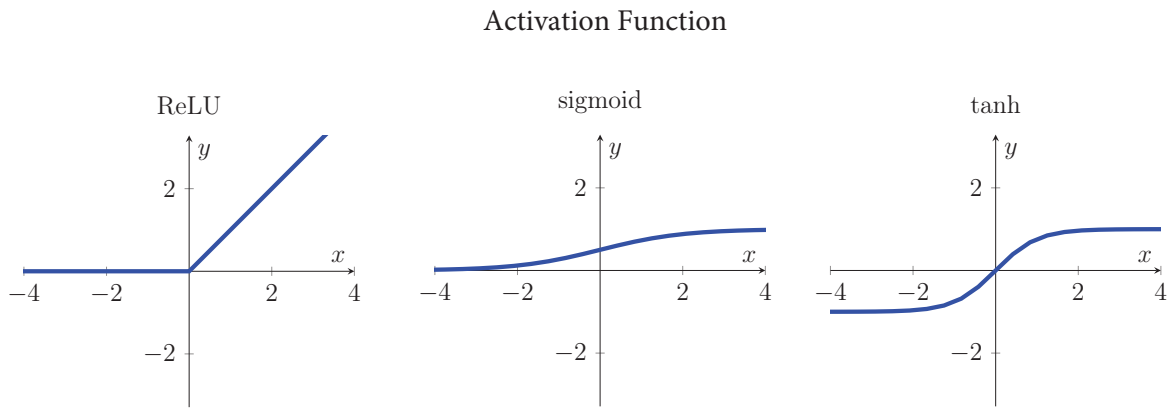



FIGURE 2.15: The graphs of the three main activation functions.

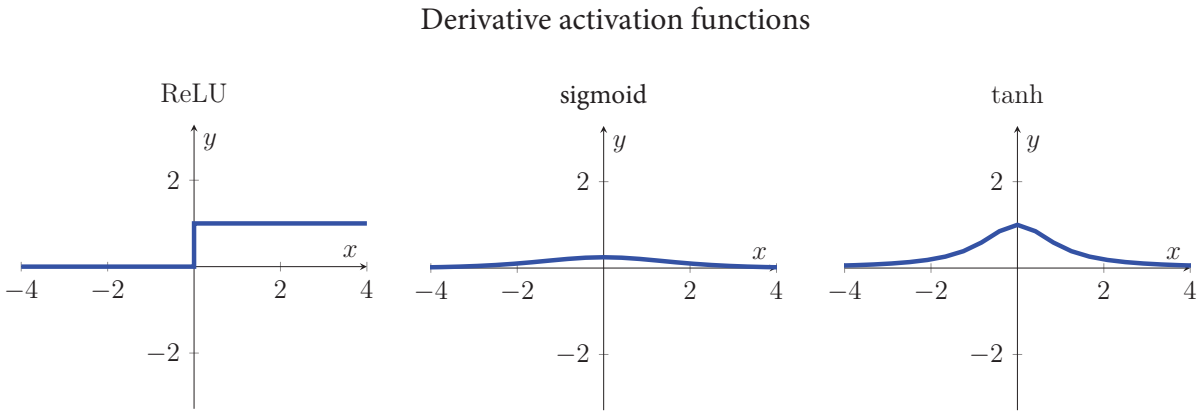



FIGURE 2.16: Derivative graphs of the three main activation functions.

**Activation function** $f(.)$ The three main activation functions in literature are as follows: $sigmoid, tanh,$ and $relu$, plotted in Figure. 2.15:

1. *sigmoid* [Neal, 1992]: When we talk about the sigmoid function, we refer by default to the logistics function:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}. \tag{2.17}$$

This function addresses the problem of the non-derivability of the Heaviside function ( Equation. 5.14). This function has two problems. Outside the range $[-3; 3]$, the *sigmoid* derivative is almost zero (see Figure. 2.16), which can prevent learning, this is known as vanishing gradient problem which is related to back-propagation nature, as the gradient of early layers is obtained by multiplying the gradients of later layers. So, if the gradients of later layers are already small (less than one), their multiplication will lead to an even smaller gradient for earlier layer, and thus the gradient vanishes very fast (gradient equals 0). Moreover, the sigmoid function does not produce a negative value. Thus, if the input consists only of positive values, the update of the weights will systematically be of the sign of the error. We will, therefore, see all weights increase or decrease when it would be appropriate to increase some and decrease others. The *tanh* activation function solves this problem.

2. Tangente hyperbolique *tanh* [Kalman and Kwasny, 1992]: The outputs of this function are zero centered which makes it easier to model inputs with negative, neutral, and positive values:

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{2.18}$$

However, similarly to Sigmoid, tanh is also susceptible to the Vanishing gradient problem.

3. Rectified Linear Units $relu$[*NairandHinton*, 2010]: It is the default and the most used function. It has several advantages.

$$relu(x) = max(0, x), \tag{2.19}$$

$$relu\,(z) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{else } \geq 0 \end{cases}$$

Its non-linearity for small values allows the normalization of its inputs. Moreover, Its derivative is very simple to compute, which allows very fast calculations and therefore helps to reduce the time of learning. However, when the input value approaches zero or equals to negative, the gradient becomes zero; thus, the network cannot perform back-propagation and cannot learn.

#### 2.2.6.3 Normalization layer

Usually, before training a neural network, we first normalize its inputs (preprocessing step). For example, inputs data can be normalized to have normal distribution (zero mean and unitary variance). This is to prevent an early saturation of the activation functions such as the sigmoid function, assuring that all data are in the same range of values.

Just as it made intuitive to have a uniform distribution for the input layer, it is beneficial to have the same identical normalized distribution for each hidden layer. Though, when we start learning, the features' distribution is continuously changing. This may slow down the training process since each layer must learn to adapt its parameter to a new distribution in every training step. Indeed, each layer's input is affected by the parameters of all the subsequent layers due to back-propagation. This problem is known as "Internal Covariate Shift", [Ioffe and Szegedy, 2015a]. The author's exact definition is *the change in the distribution of network activations due to the change in network parameters during training.*

The normalization layer is usually used after the activation layer. The most used one is the batch Normalization (BN). It was first introduced in [Ioffe and Szegedy,

2015a], where authors introduced BN as a possible solution to the "internal covariate shift". They claim that using this layer within their network can improve the results of ImageNet (2014) [Russakovsky et al., 2015] by a significant margin.

The output of the BN layer is given as follow:

$$\mathfrak{N}^{[l]} = BN(A^{[l]}), \tag{2.20}$$

where $\mathfrak{N}^{[l]}$ is a normalized activation-maps, thus we will maintain the name of *activation-map*. The BN normalizes the distribution of the output layer $y_i$ across a mini-batch of examples $\boldsymbol{x} \in \{x^{(1)}, ..., x^{(bs)}\}$, with $bs$ the number of sample in a mini-batch, to a zero-mean and a unit-variance. It operates according to the following equation:

$$\begin{aligned} y_i &= BN(x_i) = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta \\ \mu &= \frac{1}{bs}\sum_{i=1}^{bs} x_i \\ \sigma^2 &= \frac{1}{bs}\sum_{i=1}^{bs}(x_i - \mu)^2 \end{aligned} \tag{2.21}$$

where $\gamma$ and $\beta$ are two learned parameters. These parameters are used to scale and shift the normalized values, in order to make the network more flexible by allowing it to counter the normalization of the distribution. $\epsilon$ is a small number to avoid a division by zero.

#### 2.2.6.4 Pooling layer

The pooling layer, also known as a down-sampling layer consists in reducing the spatial size of the activation-maps. It allows:

1. to reduce the number of parameters, and the calculation requirements;
2. to compact the activation-maps so that the network becomes more efficient;

3. additionally, this allows minimizing the likelihood of overfitting [Jarrett et al., 2009, Krizhevsky et al., 2012].

This type of layers has no weights to updates. It is commonly used after a succession of other layers (i.e. convolutional, activation, and BN layers).

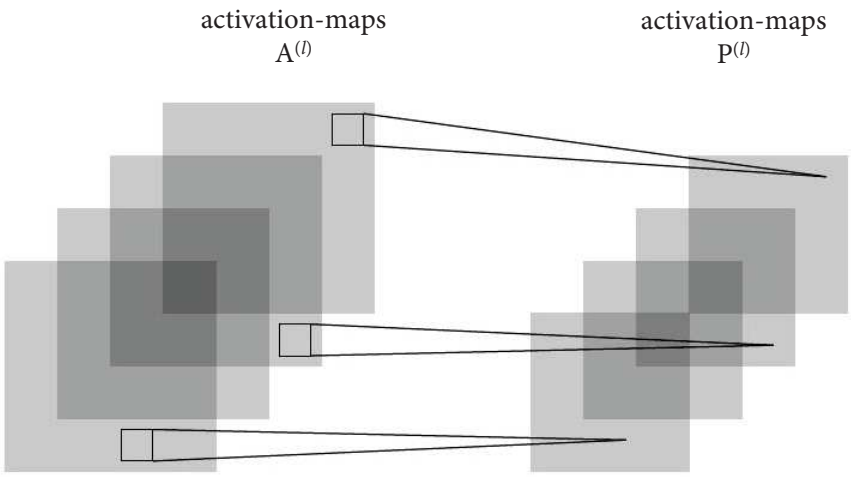



FIGURE 2.17: Illustration of a pooling layer.

A pooling layer is defined by two parameters, $p_s^{[l]}$, $W^{[l]}$ the the pooling stride, and the pooling window respectively. The size of the pooling window is $w_1^{[l]} \times w_2^{[l]}$. This layer takes as input an activation-map $\mathfrak{N}^{[l]}$ of size $n_c^{[l]} \times n_w^{[l]} \times n_h^{[l]}$, with $c_n^{[l]}$ being the number of the activation-map in the $l$-th block, and it produces an output $P^{[l]}$ of size $p_c^{[l]} \times p_w^{[l]} \times p_h^{[l]}$ where:

$$P^{[l]} = Pool_{max/avg}(\mathfrak{N}^{[l]}), \tag{2.22}$$

and its size is to be computed as follow:

$$\begin{aligned} p_c^{[l]} &= n_c^{[l]}, \\ p_w^{[l]} &= (n_w^{[l]} - w_1^{[l]})/p_s^{[l]} + 1, \\ p_h^{[l]} &= (n_h^{[l]} - w_2^{[l]})/p_s^{[l]} + 1. \end{aligned} \tag{2.23}$$

As shown in Figure. 2.17, this layer operates by sliding the window $W^{[l]}$ over each map of $\mathfrak{N}^{[l]}$ separately, and reducing the data within this windows to a single value. This operation is repeated while sliding this window by $p_s^{[l]}$ position until the entire map is spatially reduced.

The two most common methods of pooling are max and average pooling (presented in Figure. 2.18. Max pooling [Scherer et al., 2010] operates by selecting the highest value in the window region and discarding the rest of the values. The key concept of a max pooling layer is to ensure translational invariance, thus makes the network focused on feature extraction, and less sensitive to features spatial location. Put it differently, the fact that a feature is bigger or smaller, or even has a slightly different orientation, should not induce a drastic shift in the classification of the image.

The average pooling [Lin et al., 2014], on the other hand, makes it possible to consider all the values of the feature map by using the mean of the values within the window region. This kind of pooling helps the network to better generalize as it effectively combines several values into a single one, which decreases the chance of overfitting by combining several features into one. Besides these two pooling

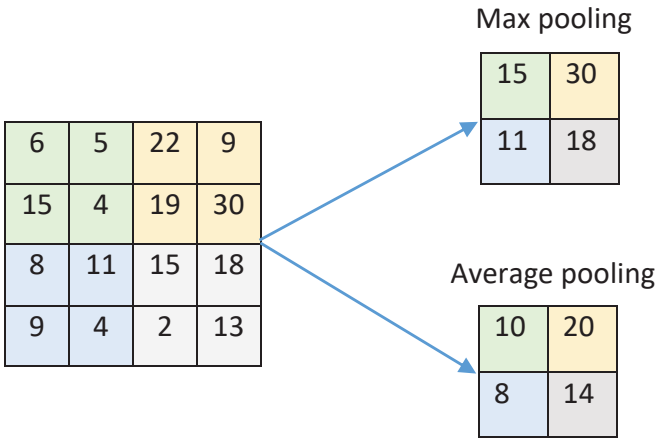



FIGURE 2.18: The two methods of pooling (max/average).

methods, we can find other methods such as the Spatial Pyramid Pooling (SPP) [He et al., 2015, Lazebnik et al., 2006, Grauman and Darrell, 2005]. This method allows having a fixed size vector at the output whatever the size of the input. This may be interesting knowing that, the input vector of a fully connected layer

must be of fixed size. This method, therefore, makes it possible to have images of variable size at the input of the network since the output feature vector will always have the same size.

**Classification module** After several convolutional blocks, the resulting feature-maps dimension is reduced to a one-dimensional vector using, either a global pooling layer (a standard pooling with windows size equal to the feature size), or an SPP layer, or simply by a flatten layer. The obtained vector is called feature-vector (equivalent to feature-vector in the two-step machine learning approach). This feature vector is the input of the classification module presented in Fig. 2.12 and designed to predict the correct class or category of the image resulting in this feature-vector [LeCun et al., 2010, 1989]. The classification module consists of a succession of one or more fully connected layer, followed by a loss function.

#### 2.2.6.5 Fully connected layer (FC)

Neurons in a fully connected layer (FC) are fully connected with neurons of the previous layer (usually two to three-layer). A set of these layers acts as an MLP (seen in Section. 2.2.3) that takes the feature-vector as input, and transforms it into the final output classes or class scores. Said differently, this fully-connected part (set of FC layers) is used to associate the different important features from the input image in order to deduce its correct class.

#### 2.2.6.6 Loss function

The fully-connected part ended with a Loss function (commonly "misnamed" as a loss layer). The Loss function specifies how training penalizes the deviation between the predicted (output) and true labels while using the stochastic gradient descent. Among the most known loss functions, we can mention, the "Softmax function", the "Mean Squared Error (L2)" and the "Sigmoid/softmax cross-entropy" function.

Now that we have talked about machine learning and deep learning techniques, we are going to talk about another technique which is intended to fool machine learning and deep learning models through malicious input. This technique, known as **Adversarial learning**, can be applied for a variety of reasons, that are discussed below.

## 2.3 Adversarial learning

In our daily life, competing against an adversarial, whether in games or competitions, make us perform better. The adversarial idea also exists in the AI domain. In deep learning field, the adversarial concept is treated in two different ways; thus, we find two researcher groups that use the adversarial idea for two different objectives. The first group main interests are **adversarial samples** and **adversarial learning**. Their objective is to treat the deep learning vulnerabilities to make the learning more robust.

*Adversarial samples* [Szegedy et al., McDaniel et al., 2016] consists in techniques used by an attacker (adversarial) to make a learned-model (i.e. ANN) unstable and behave in a way that is favourable to its benefit. To produce such unexpected behaviour, the adversary conducts a sophisticated manipulation of the input data (i.e. introducing a small noise signal to an image as shown in Figure. 2.19) to creates some tricking examples. These images confuse the model in a way that is no longer capable of classifying them (classifies the image into the wrong category with high confidence).

In a commonly cited example of adversarial samples, an altered stop sign was capable of tricking a self-driving car [Eykholt et al., 2018]. When it was simple for a human to notice the altered sign and correctly interpreting its meaning, the deep-learning model erroneously interpreted it as a 45 mph speed limit posting. In a real-world, an attack like this would resulting in the self-driving car acceleration through the stop sign, which can lead to a potentially disastrous outcome.

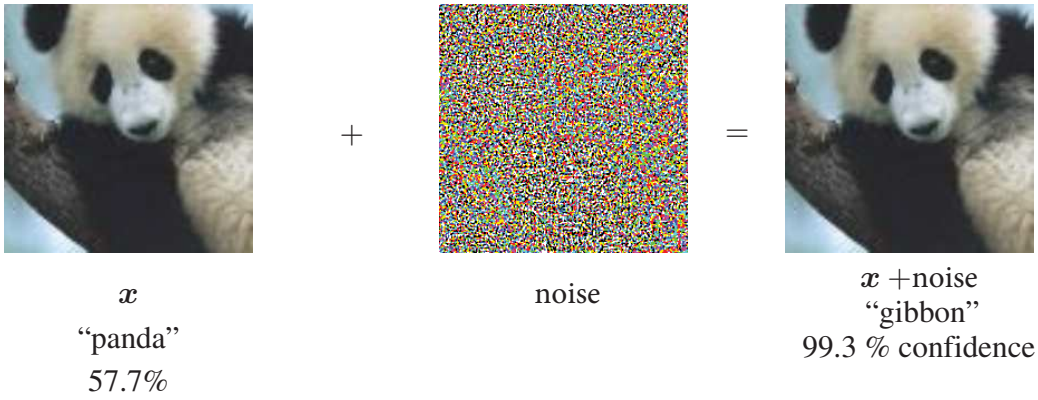



FIGURE 2.19: A demonstration of an adversarial sample, obtained by adding an imperceptibly small noise vector. The classification of the image is changed. (Extracted from [Goodfellow et al.]).

*Robust learning*, on the other hand, consists in the process of training a model (ANN) on adversarial examples, intending to make it more robust to adversarial examples attacks. So far, robust learning has primarily been applied to a small variety of problems.

The second group had a completely different view of the adversarial idea; it is seen as a stimulating factor in the learning process. The adversarial role is to help to train a model (neural network) while competing with it, which creates a self-stimulating environment for the learning process. This field of interest is known as adversarial network.

### 2.3.1 Generative Adversarial Network (GAN)

An example of an adversarial network is presented in [Silver et al., 2016], where two networks compete with each other in order to learn to play AlphaGo (learn the game state and policy). The learned model was capable of defeating a human master. Another example is the sample generator. Given a model (generator), who wants to generate images that share the same probability distribution of a dataset of real images, the adversary role is to help the generator learning how to produce realistic images. It evaluates the generated images (fake image) by comparing them with real images. The adversary evaluation is then used by the

generator in order to improve the quality of its generated images (like a student and his teacher), this is known as GANs.

The main idea of GAN is driven from the Nash equilibrium in game theory [Goodfellow, 2017]. It assumes two or more player who competes against each other. An equilibrium is obtained when each player is capable of taking the best possible move in the game, taking into account the moves of the others player.

In this context, [Goodfellow et al., 2014] have proposed a game with two players where each player is represented by an artificial neural network. The first player called the generator (noted as G) aims to generate images from random noise while respecting the distribution of a given database of real images. The second is a judge known as discriminator (noted as D). It aims to correctly determine whether a given image is a real image (from the training set) or a "fake" image generated by the generator. We give in Figure. 2.20 an illustration of a GAN architecture.

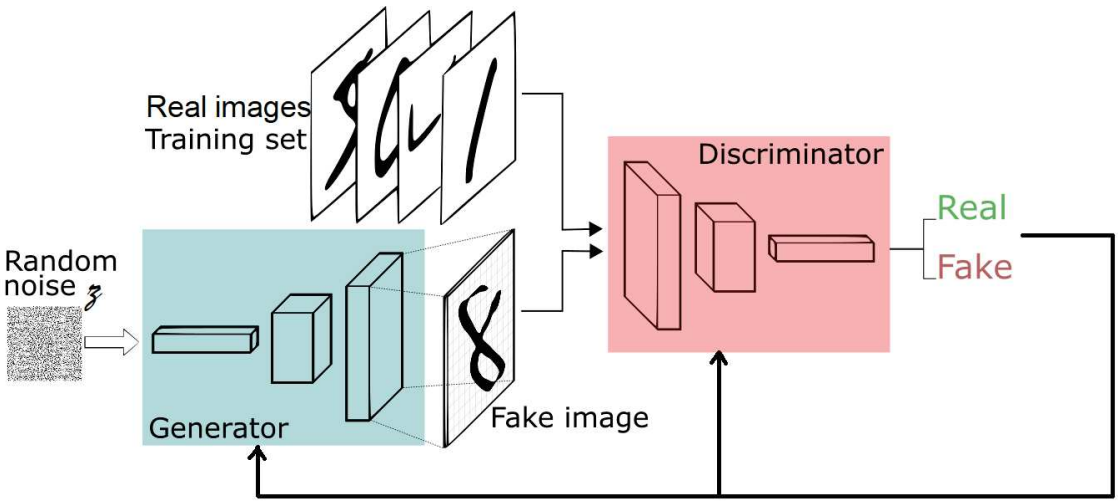



FIGURE 2.20: A general structure of a generative adversarial network (exttacted from : https://sthalles.github.io/intro-to-gans/).

In order to win the game, the two players (D, G) need to continuously optimize their parameters to improve the generation (resp. discrimination) capacity of generator (resp. discriminator). In the optimal case, the stopping criterion of the optimization process is to attain a Nash equilibrium between the two participants.

The use of GANs for generating new data is interesting for several reasons. The first is that it does not directly optimize a log-likelihood function, such as the Kullback-Leibler distance (cross-entropy) [Stemmer et al., 2002], between generated images and real images because this would be difficult to exploit (the dimension of the image space is too large to even indirectly access to the real probability of the images). Instead, the gradient is calculated by the intermediary of the discriminator whose performances evolve over time according to all examples in the database. Besides, confronting an opponent introduces free supervision since data are simply noted as real/fake.

### 2.3.2 Training of a Generative Adversarial Network (GAN)

G and D are parameterized by neural networks where each network competes against the other. Thus the training of such networks is considered as a two-player minimax game. The intent of the discriminator (D) is to maximize its accuracy of discriminating real image $\boldsymbol{x}$ from the generated images. On the other side, the generator's goal is to generate images the most realistic possible, and thus ensures that the discriminator makes more classification error.

Like any regular ANN, back-propagation is used to train the network parameters, but the fact that there are two networks makes the use of back-propagation slightly different. More precisely, the used loss functions and the number of iterations (the training is altered between each model) are two major components where the GANs differ. For the discriminator, the used loss function will be nothing more than a regular binary cross-entropy loss. Other loss functions may be used such as Wasserstein distance, Maximum Mean Discrepancy, ..etc.

$$-\left(y \log(\hat{y}) + (1-y) \log(1-\hat{y})\right) \tag{2.24}$$

In this context, $y = 1$ if the image fed to the discriminator is a real image where $y = 0$ if it is a fake one. $\hat{y}$ is the probability that the image is a real image,

where $(1 - \hat{y})$ is the predicted probability of the input image being a fake image. In Equation. 2.24 the probability $\hat{y}$ is the prediction output of the discriminator, thus it can be represented as $D(\boldsymbol{x}^{(i)})$ with $\boldsymbol{x}^{(i)}$ being the $i^th$ input image of the database. Equation. 2.24 then looks like below:

$$- (y^{(i)} \log(D(\boldsymbol{x}^{(i)})) + (1 - y^{(i)}) \log(1 - D(\boldsymbol{x}^{(i)}))) \tag{2.25}$$

Taking a close look at Equation. 2.25, one can notice that for real images $\boldsymbol{x}$, the equation is resumed in its first part since the second part will be zero. As for fake images, it is exactly the reverse. Keeping this in mind, the image $\boldsymbol{x}$ in the second part can be replaced by $G(\boldsymbol{z})$ since fake image represents the output of the generator $G$, with $\boldsymbol{z}$ being a random noise vector that feeds the generator. The discriminator loss function is given as follow:

$$L_D = -(y^{(i)} \log(D(\boldsymbol{x}^{(i)})) + (1 - y^{(i)}) \log(1 - D(G(\boldsymbol{z})))) \tag{2.26}$$

However, in the GAN paper [Goodfellow et al., 2014], the Equation. 2.26 was given as follow:

$$L_D = (y^{(i)} \log(D(\boldsymbol{x}^{(i)})) + (1 - y^{(i)}) \log(1 - D(G(\boldsymbol{z})))) \tag{2.27}$$

were the first sign was flipped so that the updating of the discriminator weight can be done by "ascending" its stochastic gradient.

As for the generator $G$, its loss function $L_G$ is designed to make the discriminator less confident of his prediction, which is equivalent to a maximization of the discriminator loss function $L_D$ . However, the first part of Equation. 2.26 is meaningless to the generator (we do not want that the discriminant mispredicts the real images), so only the second part should be maximized. In other words, the loss function of G will be the same as D's loss function with the first term ignored,

and the sign flipped. As indicated in the GAN paper [Goodfellow et al., 2014], we can just ignore the sign flipping, and instead of updating generator weight by "ascending" its stochastic gradient, it is updated by "descending" its stochastic gradient, the generator loss is then giving as follow:

$$L_G = \log(1 - D(G(\mathfrak{z}))) \tag{2.28}$$

The training procedure consists of $k$ iteration of alternative training. First, we fix the parameters of $G$ and optimize $D$ parameters for $k_D$ iteration through maximization of $L_D$ of equation. 2.27. Then, we fix $D$ and optimize $G$ by minimizing $L_G$ in Equation. 2.28 for $k_G$ iteration.

We stop the training of the GAN when we attain a Nash equilibrium, or when we reach the maximum number of iterations.

## 2.4 Conclusion

In this chapter, we have focused on machine learning techniques used for classification purposes. We have first presented in Section. 2.1 the two-step learning approach, which consists in a feature extractor and a trainable classifier. Feature extractor extracts pertinent features, which are then fed to the trainable classifier to learn a separable function. We have, therefore provided a brief overview of how these approaches process, then we have highlighted the limits of such approaches.

We, therefore, discussed the second category of approaches, the neural networks that automatically extract features by learning from examples in Section. 2.2. We started by presenting the elementary component (neuron), the architecture and the training method. In Section. 2.2.6, we focused on convolution neural networks since they are more adapted to process images.

In Section. 2.3, we have presented a totally different family of deep-learning approach which is the generative model, where the objective is not only to discriminate data but also to generate new samples that share the same probability distribution as real images.

# Chapter 3

# Steganography in spatial domain

## Contents

## 3.1 General presentation

In all histories of civilization, whether in wars, business or politics, information was a decisive factor. Those who had access to critical information could be empowered to win any challenge. That is why exchanging secret information has always been a hot topic. This has led to the emergence of private communication, such as cryptography and steganography. Cryptographic techniques secure information access through an encryption process, thereby making it unintelligible to those who do not have the necessary rights. But in most cases, communication with encrypted messages can attract attention. Especially knowing that the communication channel can be monitored by a third party who analyzes all messages passing through the communication channel. In such a case, the only feasible solution for both parts to communicate is to hide the very fact that they are communicating. This may be possible using steganography.

Steganography, or the science of secret communication, is the art of hiding a secret message within another carrier (text, image, sound, video...) of an innocuous nature so that the very existence of the secret message is hidden from view. In other words, in steganography, we want to make it difficult, or even impossible, for a third-party to distinguish between an innocent carrier and a carrier holding a secret message.

The first reference to the term "steganography" was found in Johannes Trithemius Steganographia's book in 1499. Although steganography techniques had existed a long time before, at the time, it was the story of what could be called ancient steganography, i.e. methods of dissimulating information using, for example, invisible ink on paper, or simple mechanical aids [Fridrich, 2009].

Recently, with the rise of the digital world, digital media has become a practical communication medium carrier. This is what gave birth to the *modern steganography* [Pfitzmann, 1996]. The purpose of modern steganography is to hide a secret message, of relatively large size, in a digital media called *cover*, so that the resulting media, called *stego*, remains, at least to the naked eye, identical to the cover.

This means that the existence of the secret message in the stego is imperceptible and practically undetectable by an eavesdropper. The secret message can be anything that can be converted into a bit stream, plain text, encrypted text or even an image. As for cover media, many different types are available, such as image, sound, video, text, etc. Nevertheless, images are the most commonly used media in steganography, as they are very commonly exchanged over the Internet, making it an excellent carrier for hiding secret information.

We give in Figure. 3.1 a simple representation of a modern stenographic system during the embedding and the extraction procedure.

- **Embedding process:** The sender, Alice, embeds the secret message into an innocent digital media using a shared-secret key [1], the resulting stego media is then transmitted via a public channel (e. g. Internet) to the recipient Bob.
- **Extraction process:** The receiver, Bob, uses the shared-secret key, retrieves the secret message from the received stego media.

However, these tasks can prove very challenging given the existence of an eavesdropper, Eve, who examines all data that passes through the communication channel. The eavesdropper should not be aware of the existence of communication between Alice and Bob. Otherwise, she will break the communication channel.

In this document, we focus mainly on *spatial steganographic* [2] schemes that use digital grayscale images as a transmission medium.

## 3.2 The three families of the steganography

In steganography, three main families can be found. These families were proposed based on how we embed the secret message [Fridrich, 2009]; there is

[1] The secret key is shared between the sender and the recipient prior to communication.

[2] Spatial domain referring to the fact that the pixel values are used to embed a message (by opposition to frequency domain resulting of the Fourier transform of the image).

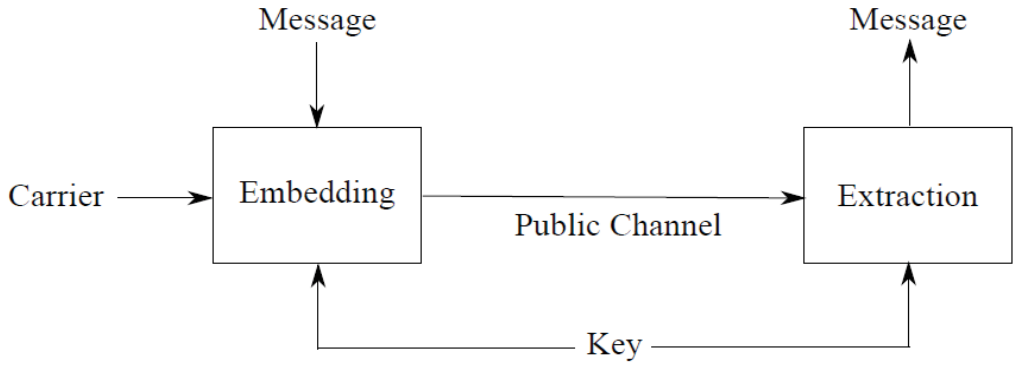



FIGURE 3.1: The embedding and extraction framework of a steganographic scheme.

1. the steganography by cover selection,
2. the steganography by cover syntheses,
3. the steganography by cover modification.

In the following, we will present these three families of steganography, highlighting their advantages and drawbacks.

### 3.2.1 Steganography by cover selection

In steganography by cover selection, the sender Alice has a fixed database of images beforehand. From this database, Alice selects the one that best fit the desired message. For example, Alice can transmit a bit of information to Bob simply by considering the time of day the image was taken (day or night). Similarly, the presence or not of a bird within the image sent may have a hidden signification shared only between the sender and receiver, such as whether or not we escape tomorrow.

An important case of steganography by cover selection is the use of a Message-digest (hash) function where Alice uses a hash function, with a secret key shared with Bob, to transmit her message. In such a case, Alice scans her image database using message digestion functions with the objective of finding an image whose digest corresponds to the desired message bitstream. Once found, the image is

forwarded to Bob, who can easily read the secret message by reapplying the hash function with his secret key. One can notice that this method becomes very quickly impracticable in reality because the number of attempts required to obtain a match can be impracticably high because it depends exponentially on the length of the digest.

The advantage of such approaches is that they are almost undetectable [Fridrich, 2009]. Indeed, since the cover does not undergo any modification, it is impossible to guess that there is a hidden message. However, the major problem with these methods remains its very limited embedding capacity.

### 3.2.2 Steganography by cover synthesis

In steganography by cover synthesis, instead of choosing a cover from an existing database, Alice would simply create such a cover that best conveys the secret message. If Alice is able to create a cover, with a known distribution between her and Bob, she will be able to hide safely its secret message.

A real example of steganography by cover synthesis, in which text is used as cover instead of digital images, is a program [3] using the so-called mimic function [Wayner, 1992]. The goal of this program is to encode the secret message so that it looks like a spam message.

The benefit of such approach is that it offers a high-security level. However, it is very limited in term of embedding capacity, which makes it a less interesting approach [Fridrich, 2009].

### 3.2.3 Steganography by cover modification

Steganography by cover modification is the most widely used and studied steganography paradigm. Its principle consists in modifying a (already existing) cover object to hide a secret message with the help of a shared-secret key while trying to

[3] http://www.spammimic.com

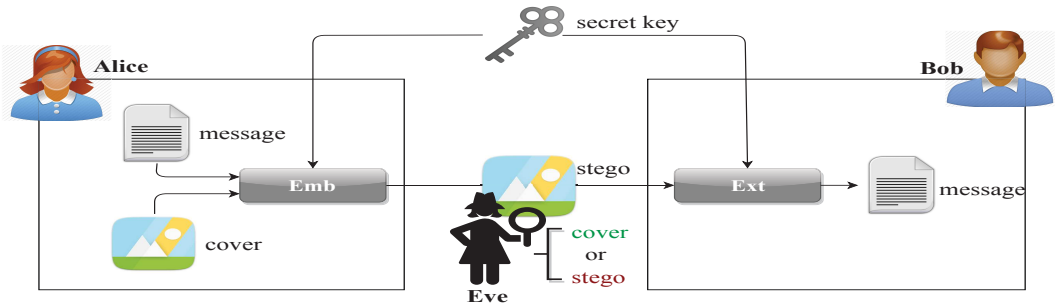



FIGURE 3.2: General diagram of steganography.

preserve "as much as possible" the original statistics of the used cover.

We illustrate in Figure. 3.2 the general principle of steganography by cover modification. Considering the following notation:

- $\mathcal{C}$ a set of cover objects,
- $\mathcal{K}$ a set of secret-key shared between both Alice and Bob,
- $\mathcal{M}$ a set of messages,
- $\mathcal{S}$ a set of stegos objects.

The embedding and extraction represented by Alice and Bob can be formulated as follow:

$$\begin{aligned} Emb &: \mathcal{C} \times \mathcal{M} \times \mathcal{K} \to \mathcal{S}, \\ Emb(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{k}) &= \boldsymbol{s}, \\ Ext &: \mathcal{S} \times \mathcal{K} \to \mathcal{M}, \\ Ext(\boldsymbol{s}, \boldsymbol{k}) &= \boldsymbol{m}. \end{aligned} \tag{3.1}$$

with , $\boldsymbol{c} \in \mathcal{C}$, $\boldsymbol{m} \in \mathcal{M}$, $\boldsymbol{k} \in \mathcal{K}$

$Emb$ and $Ext$ are the embedding and extracting mapping functions respectively. $\boldsymbol{c} \in \mathcal{C}$ is a cover media, $\boldsymbol{m} \in \mathcal{M}$ is a secret message embedded in $\boldsymbol{c}$ using $\boldsymbol{k} \in \mathcal{K}$, where $\boldsymbol{k}$ stands for the secret key shared between Alice and Bob in the prisoner

problem [Simmons, 1984], $\boldsymbol{s} \in \mathcal{S}$ is the stego obtained by embedding. In this manuscript, we focus on steganography by cover modification methods.

By taking a look at steganography methods that exist in the literature, we can find two ways of designing a good steganographic algorithm:

1. By defining a cover image model that will be preserved during the embedding process [Sallee, 2003, Kodovský and Fridrich, 2008].

   **Statistical model-preserving steganography** relies on preserving (as well as possible) a particular type of a cover image model that was previously defined. These methods are undetectable and highly secured as long as the defined model describes perfectly the covers. Nevertheless, these methods can be detected by an attacker (steganalyst) who works outside the defined model by identifying a quantity that is not perfectly preserved during the embedding.

2. By using content-adaptive embedding which relies on minimization of embedding distortion function, presented in section 3.3.1, which is usually defined heuristically. As an example of these methods we have, HILL [Li et al., 2014], UNIWARD [Holub et al., 2014], WOW [Holub and Fridrich, 2012], HUGO [Pevný et al., 2010b].

## 3.3 Adaptive Steganography

Since the STC coding proposal [Filler et al., 2010], giving the possibility to code a message by taking into account the content of the image (i.e. adaptive coding) and the first algorithm uses this coding HUGO [Pevný et al., 2010b] in 2010, adaptive steganography has been introduced as the most secure approach.

It consists in embedding the secret message while trying to introduce the smallest possible distortion into the cover image. Said differently, adaptive steganography algorithms attempt to embed the secret message in the areas of the cover image

that are difficult to detect. Since steganography by a cover modification disrupts the cover statistics during the message embedding, it is, therefore, necessary to select secure areas of the cover which, even when modified, do not have a significant impact on the overall cover statistics. This is achieved by most current adaptive methods using a distortion function that models the embedding impact on security (presented in Section. 3.3.1). Compared to model-preserving method [Kodovský and Fridrich, 2008], adaptive steganography methods offer a better generalization and even a better security level. Besides, it enables the development of steganographic algorithms driven by the performance, in term of security, against a steganalyser.

Adaptive steganography algorithms, just like other *steganography by cover modification* algorithms follow the following three steps:

1. determine which positions are best for the embedding,
2. encode the message,
3. embed the message.

Each of these is detailed in the following sections as well as the way to put them together to create an "adaptive embedding algorithm".

### 3.3.1 Distortion measure and cost map

Modern or adaptive steganography algorithms, as indicated before, aim at embedding a given message into a cover, while trying to minimize the impact induced by the embedding process [Fridrich and Filler, 2007]. In order to achieve this objective, it is important first to establish a distortion measure capable of quantifying the statistical detectability due to embedding. Before proceeding with the cost map explanation, it is essential at this point to have first a better understanding of what is a distortion measure.

Distortion is a measure used by a steganographer to model the overall embedding-impact induced by the message embedding process. Adaptive steganographic schemes embed a secret message according to the minimization of a function for the purpose of preserving the cover medium as much as possible. It is modelled by a mathematical function as follow,

$$D : \mathcal{C} \times \mathcal{S} \rightarrow [0, \infty[, \tag{3.2}$$

In the case of grayscale image the Equation. 3.2 is given as:

$$D : \{0, ...255\}^n \times \{0, ...255\}^n \rightarrow [0, \infty[ \tag{3.3}$$

Where $D$ reflects the total distortion, it should give an approximation of the statistical detectability caused by the embedding of a secret message.

The embedding impact can be modelled by a function that measures the distance between the cover image and the stego image in feature space or another descriptive space [Ker et al., 2013]. Let $\boldsymbol{x} = (x_1, ..., x_n) \in \mathcal{C}$ with $x_i \in \{0, ..., 255\}$ a cover image composed of $n$ elements, and $\boldsymbol{y} = (y_1, ..., y_n) \in \mathcal{S}$ with $y_i \in \{0, ..., 255\}$ the associated stego image. $\boldsymbol{m}$ is a secret message composed of $m$ elements where $\boldsymbol{m} = (m_1, ..., m_m) \in \{0, 1\}^m \in \mathcal{M}$. Then, the distortion function $D$ can be defined as follow:

$$D(\boldsymbol{x}, \boldsymbol{y}) = ||f(\boldsymbol{x}) - f(\boldsymbol{y})||, \tag{3.4}$$

with $f(\boldsymbol{x}), f(\boldsymbol{y})$ the features vector that describes the cover, and stego, respectively. Usually, the function $f$ returns a vector composed of real values. These real values are often calculated using the occurrence of pixel values [Ker et al., 2013].

The distortion function in its non-additive and non-local form given in Equation. 3.4 presents a real problem for the steganographer since it is too complex finding a

code minimizing it. In order to overcome and simplify the problem, two approaches have been proposed.

The first approach is to make the hypothesis that the modification of a pixel does not affect the detectability of neighbouring pixels so that the distortion can be approximated in an additive version. The authors in [Filler et al., 2010] and [Filler et al., 2011], among other, adopt this perspective. They propose an additive version of the distortion which is based on the use of the so-called **cost map** $\boldsymbol{\rho} = \{\rho_i \in [0, \infty[\}_i^n$. It consists in assigning to each pixel of the cover $x_i$ a value $\rho_i \in [0, \infty[$. This value reflects the impact of the modification of the $i^{th}$ pixel on the global security. It is defined as in Equation. 3.5 [Holub et al., 2014]. The case where $\rho_i = \infty$ means that the pixel $x_i$ is not allowed to be modified, known as a *wet pixel*.

$$\rho_i = D(\boldsymbol{X}, \boldsymbol{X}_{\sim}\boldsymbol{X}_i), \tag{3.5}$$

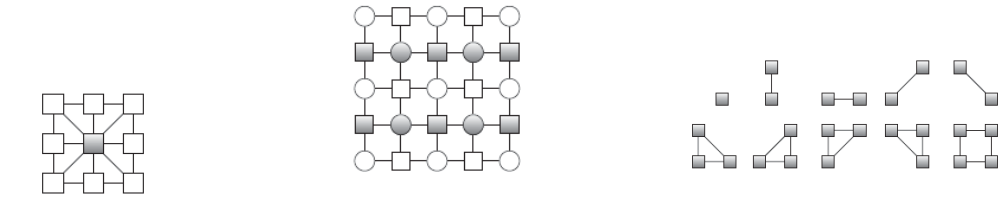
where $\boldsymbol{X}_{\sim}\boldsymbol{X}_i$ is the cover image whose $i$-th pixel is modified.

With the hypotheses of $\rho_i^+ = \rho_i^-$ which means that the modification cost of a pixel $x_i$ is the same regardless of whether it is modified by $-1$ or $+1$, the equation in. 3.4 can be then approximated to a an additive version using the cost map as follow:

$$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} \rho_i \left| x_i - y_i \right|, \tag{3.6}$$

where $|.|$ is the absolute function.

As mentioned before, the objective of adaptive steganography is to minimize the impact induced by the embedding process, which is measured using a defined distortion function such as Equation. 3.6. As it stands, we can quickly conclude that the major challenge of this approach is the computation of the cost map $\boldsymbol{\rho} = \{\rho_i \in [0, \infty[\}_i^n$.



(a) 3×3 Neighbourhood

(b) Subdivision of a set of pixels into four separate subgroups $S = \{S_1 \cup S_2 \cup S_3 \cup S_4\}$, with $S_1$:white circles, $S_2$: gray circles, $S_3$:white squares, et $S_4$ : gray squares

(c) Set of possible clicks for neighbourhood of $3 \times 3$

FIGURE 3.3: Example illustrating 1) an image partitioning, 2) the click system [Kouider, 2013].

How computing the $\rho_i$ scores to measure the impact of the embedding process on security remains an open question, which began to be studied at the end of 2010, through the BOSS competition [Bas et al., 2011] using the HUGO algorithm [Pevný et al., 2010b].

In later sub-section 3.4 we will present the current state-of-the-art algorithms, and how they derive the cost map in the context of additive distortion.

The second approach, in order to minimize $D$ from Equation. 3.6 consists in dividing the global problem into simple local sub-problems that may be relatively easier to solve. In [Filler and Fridrich, 2010] authors have proposed a new non-additive distortion function. It is defined as a sum of local distortion functions calculated on a particular neighbourhood system. Their approach operates by grouping the cover pixels in sub-lattice of pixels where each pixel is independent of other pixels. These sub-lattice are noted as $\mathcal{S} = \{\mathcal{S}_1 \cup ... \cup \mathcal{S}_n\}$. The message is then split into small parts where each part has to be embedded in a sub-lattice while minimizing a distortion measure. The used distortion function is a sum of local distortions calculated on cliques, where a clique is sub-lattice where each pair of its different elements are neighbours, an example is given in Figure. 3.3. More details can be found in [Filler and Fridrich, 2010].

Now that we have seen how to define a cost map and the distortion to be minimized, we will see in the following how to encode our message.

### 3.3.2 Syndrome coding

#### 3.3.2.1 Error detection and correction code

Error-correcting codes have their roots in a very concrete problem related to data transmission. In the vast majority of cases, data transmission is made using a communication channel which is not entirely reliable. In other words, data, when circulating on such channel, are likely to be altered. For instance, we can cite the example of digital communication via the Internet of a digital medium (image, sound, text file, etc.). During transmission, the transmitted digital media may be altered due to the interference of noise that may be present in the transmission channel, i.e. bits may be switched from 0 to 1 or inversely. The meaning of the message can be altered or even damaged. Therefore, it is important to find a way to make the transmission of these data more reliable.

With the aim of correcting the errors that may occur during data transmission, or even during read/write to a physical medium such as hard disk, DVD, ..etc., error-correcting codes were developed. These codes proceed as follows: first, check the integrity of the intercepted data, then correct, as far as possible, the errors produced during transmission.

For a better understanding of the coding method, we provide the following simple example. The sender (an encoder) wants to send a message that is presented as a bit sequence ($b_1b_2..bn$). To this end, the encoder reports 3 times the message bits as follows ($b_1b_1b_1, b_2b_2b_2..bnbnbn$). The encoded message is then transmitted to the receiver (decoder). On the receiver side, and in order to detect possible errors, the decoder can simply compare the received triplet of bits. If the bits of the triplet are different, then an error has occurred during the transmission. Now, Considering that there is at most one error for each 3-bit sequence, to correct the error, the decoder has to choose the symbol that appears twice in each received triplet. However, the presented example is only a toy example, many more effective and sophisticated codes have been proposed in the literature [Pach, 2007]. One of the most studied and most common error correcting-codes is liner code.

### Linear Error-Correcting Codes

A linear code is used to match each information word to a code word, by using a linear function. It facilitates the construction of the code as well as the control of received messages.

A linear code $\mathfrak{C}$ is described by three parameters: $[n, k, \delta]$, with $n > k$. $n$ refers to the length of the code, $k$ represents the size of the code, corresponding to the size of the words once decoded. $\delta$ defines the minimum distance between each word in the code. The encoding application is linear. It is therefore calculated and represented thanks to its generator matrix. A code is entirely defined by its generator matrix.

**Generator matrix** It is a matrix whose rows form a basis for linear code. To transmit a message $\boldsymbol{m} \in \mathcal{F}_2^k$, -with $\mathcal{F}_2$ the finite field with 2 elements- through a noisy channel, it is first transformed into a code word $\boldsymbol{c} \in \mathcal{F}_2^n \in \mathfrak{C}$, using code's generator matrix, noted $\mathbf{G} \in \mathcal{M}_{n,k}(\mathcal{F}_2)$ (i. e. with $n$ rows, $k$ columns). The messaging encoding operation is performed by the following equation:

$$\boldsymbol{c} = \mathbf{G} \cdot \boldsymbol{m}. \tag{3.7}$$

If the matrix $\mathbf{G}$ is of the form $(\boldsymbol{I}_k; \mathbf{A})$ where $\boldsymbol{I}_k$ denotes the $k \times k$ identity matrix, and $\mathbf{A}$ is a redundancy matrix, we say that $\mathbf{G}$ is in standard form.

**Parity-check matrix** The main interest in considering linear codes is that they do dispose of effective decoding algorithms, thanks to the use of the Parity-check matrix. The receiver can check whether the received word $\boldsymbol{c'} \in \mathcal{F}_2^n$ is a code word. To that purpose a syndrome $\boldsymbol{s} \in \mathcal{F}_2^{n-k}$ is computed using the so-called parity-check matrix $\mathbf{H} \in \mathcal{M}_{n-k,n}(\mathcal{F}_2)$:

$$\boldsymbol{s} = \mathbf{H} \cdot \boldsymbol{c'}. \tag{3.8}$$

The syndrome $\boldsymbol{s}$ is the result of the received codeword on a parity check matrix to determine whether it is error-free or not. If the syndrome elements equal to zero

($\boldsymbol{s} = (0, ..., 0)$), the receiver infers that the received word is a code word and that no error has occurred during the transmission ($\boldsymbol{c'}$ is error-free). Otherwise, the received word contains one or more bit errors that occurred during transmission.

#### 3.3.2.2 Error correcting code for steganography

In [Crandall, 1998], the author has proposed for the first time to model the message embedding/extracting steps using error-correcting codes. This model called syndrome coding or matrix embedding. Despite the fact that this model was proposed in 1998, it has been made popular -in 2001- by its use in the design of the F5 algorithm using Hamming codes [Westfeld, 2001]. This has paved the way for the appearance of more sophisticated codes which are used by current state-of-the-art steganography algorithms.

Syndrome coding methods use the error-correcting codes to transform the transmitted message into a syndrome. The sender must determine how to modify the cover image, so that the syndrome calculated at the receiver's side corresponding to the desired message, and so that the image is the least altered.

Let suppose that $\boldsymbol{x} = \{(x_1, ..., x_n) \in \{0, ..., 255\}^n$ is a cover image, the sender objective is then to find a stego $\boldsymbol{y} = \{(y_1, ..., y_n) \in \{0, ..., 255\}^n\}$ where:

$$\begin{aligned} \boldsymbol{y} &= lsb_x(\boldsymbol{x}, \boldsymbol{v}), \\ \mathbf{H} \cdot \boldsymbol{v} &= \boldsymbol{m}. \end{aligned} \tag{3.9}$$

with $\boldsymbol{v} = (v_1, ..., v_n) \in \{0, 1\}^n$ being the modification vector, and H the parity-check matrix. $lsb_x$ is the embedding method, which we will discuss later in Section. 3.3.3). We give in the following some examples of the matrix embedding. Matrix embedding methods are used to find a solution to Equation 3.9 by mapping $\boldsymbol{m} = (m_1, ..., m_m) \in \{0, 1\}^m$ to $\boldsymbol{v} = (v_1, ..., v_n) \in \{0, 1\}^n$.

### Hamming codes

Thanks to their very particular properties, this family of linear codes allows both error detection and correction, for maximum error. Among the properties that make this code very interesting is that its minimum distance $\delta$ is equal to 3. The Hamming codes are are defined by an integer $p \in \mathbb{N}^+$, such as $n = 2p - 1$ and $k = 2^p - 1 - p$.

Hamming codes are the first correcting codes to have been used for the matrix embedding concept, and the first implementation was with F5 algorithm [Westfeld, 2001]. This algorithm, among with other embedding algorithms which use hamming code, consider that the embedding could take place in an equivalent way in each element of the host sequence. In terms of cost maps, this is equivalent to having a detectability map in the form $\boldsymbol{\rho} = \{\rho_i = 1\}_{i=1}^{n}$.

### Wet paper codes

This family of codes is another effective solution that allows the communication of secret information. It consists in preventing the use of certain elements of the host sequence to ensure a high undetectability (these elements are called "wet" and cannot be written on). By using the notion of a cost map, we can say that the cost map is a two-valued map. $\boldsymbol{\rho} = \{\rho_i \in \{0, \infty\}\}_{i=1}^{n}$. An example of an algorithm using this approach is nsFS The reader can have further details on how to use the wet paper code for steganography in [Iranpour and Safabakhsh, 2015, Fridrich et al., 2005].

### Syndrome Trellis Codes (STC)

In 2010, a new code was proposed taking into account a more subtle and adaptable detectability map, with $\boldsymbol{\rho} = \{\rho_i \in [0, \infty[_{i=1}^{n}\}$. This is the so-called STC approach, for "Syndrome Trellis Codes" [Pevný et al., 2010b]. We report here the codes description as presented in [Filler et al., 2010, 2011]. The STC are codes whose decoding algorithm (i.e. Viterbi decoding) is based on a trellis algorithm. Let $\boldsymbol{m} = (m_1, ..., m_m) \in \{0, 1\}^m$ be a secret message, we want to find the vector $\boldsymbol{v_s}$, such that during of the reception of $\boldsymbol{m}$, the syndrome $\boldsymbol{s}$ is defined as:

$$\boldsymbol{s} = \mathbf{H} \cdot \boldsymbol{v_s} = \boldsymbol{m}, \tag{3.10}$$

where $\boldsymbol{v_s}$ represent the stego vector.

To this end, the trellis approach uses a check-parity matrix $\mathbf{H}$ obtained by filling the diagonal of a sparse matrix with a set of sub-matrix $\hat{\mathbf{H}}$ of size $h \times w$ together. The $n$ copy of sub-matrices $\hat{\mathbf{H}}$ are placed one next to the other and shifted one line down, as shown in Figure. 3.4. The rest of the matrix is set to 0.

In order to find a stego vector $\boldsymbol{v_s}$ associated to the stego image with the smallest distortion, STC uses a trellis that includes corresponding vertices to the possible values for $\boldsymbol{v_s}$, while the edges hold the costs given in the cost map (seen previously in Section. 3.3.1).

The graph is navigated with the purpose of finding a code word $\boldsymbol{v}$ that satisfies $\mathbf{H} \cdot \boldsymbol{v_s} = \boldsymbol{m}$. Each found vector $\boldsymbol{v}$ is represented as a path through the trellis that goes from the leftmost to the right(end) of the trellis. Paths are created, continued or stopped according to the costs from the given cost map. The optimal solution is given by the lowest cost path (the shortest path). Currently, the trellis approach is the most effective practical approach in terms of embedding efficiency; it is the closest method to the theoretical boundary obtained via a simulator (see Section. 3.3.4). The example in Figure. 3.4 is a simplified illustration of the workings principle of the trellis approach (STC).

Note that these codes are used in almost all modern steganography algorithms such as Hugo [Pevný et al., 2010b], S/J/SI-UNIWARD [Holub et al., 2014], HILL [Li et al., 2014], MVGG [Sedighi et al., 2015], [Holub and Fridrich, 2012], [Kouider et al., 2013] ..etc.

Now that we have seen how to associate cost to pixels in Section. 3.3.1, and how to encode a message in Section. 3.3.2.2, we discuss in the last part the **embedding**.

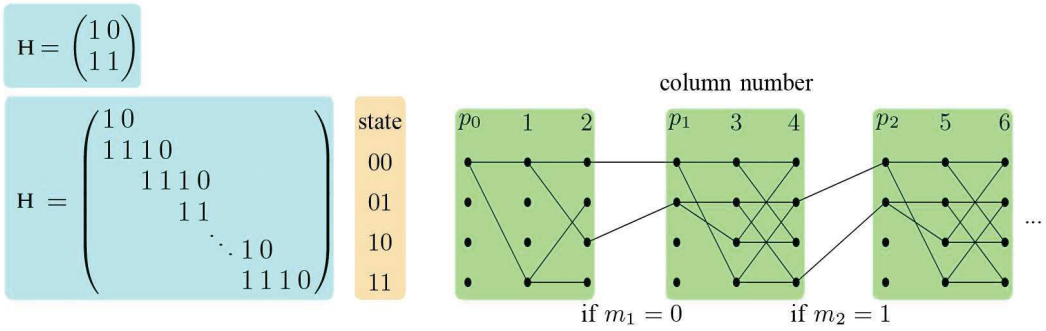



FIGURE 3.4: Illustration of the functioning principle of the trellis approach (STC). **H** is the check-parity matrix. The graph is navigated from left to right during the message embedding, gradually exploring the relevant possibilities for the choice of the stego vector.

### 3.3.3 Embedding

Message embedding (after application of syndrome coding) is achieved by modifying the pixels of the cover image $\boldsymbol{x}$. For this purpose, we proceed as follows:

1. $\boldsymbol{x} = (x_1, ..., x_n) \in \{0, ...255\}^n$ is the cover image (grayscale image),
2. consider the least significant bits (LSB) of $\boldsymbol{x}$'s pixels to generate the so-called cover vector and noted as $LSB(\boldsymbol{x}) = \boldsymbol{v_c} = (v_{c1}, ..., v_{cn}) \in \{0, 1\}^n$,
3. use the shared secret key (between both Alice and Bob) to shuffle $\boldsymbol{v_c}$ and the associated cost map $\rho$ (as explained in sub-section. 3.3.1). noted $\boldsymbol{v'_c}$ and $\boldsymbol{\rho'}$.
4. use STC (presented in Section. 3.3.2.2) to generate the stego vector $\boldsymbol{v_s} = (v_{s1}, ..., v_{sn}) \in \{0, 1\}^n$. STC is fed with cover vector $\boldsymbol{v_c}$, cost map , and the message $\boldsymbol{m}$,
5. embed $\boldsymbol{v_s}$ within the cover image $\boldsymbol{x}$, in order to obtain the stego image $\boldsymbol{y}$.

In step 5, the stego vector is embedded within the cover image by modifying some selected pixels. These selected pixels are modified whether by making their LSB correspond to the stego vector (LSB replacement) or by adding/subtracting

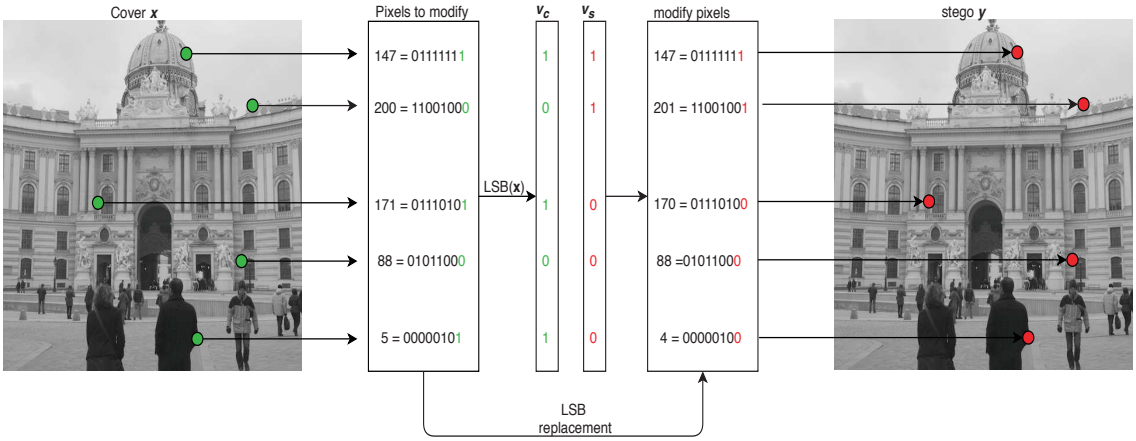



FIGURE 3.5: Illustration of the embedding algorithm in the case of an LSB replacement. The cost map (3.3.1) as well as the key will determine a sequence of pixels to be modified in the cover image $\boldsymbol{x}$. It is from these pixels that we generate the cover vector $\boldsymbol{v_c}$ through the extraction of the LSBs. A syndrome coding technique is applied to transform the vector $\boldsymbol{v_c}$ into a stego vector $\boldsymbol{v_s}$ (explained in Section. 3.3.2.2). Finally, the stego image is generated by modifying the value of the selected pixels in the cover image $\boldsymbol{x}$ so that their LSB correspond to the stego vector $\boldsymbol{v_s}$.

"1" from the pixel's value when the stego vector element is different from the corresponding cover vector element (LSB matching).

### Steganography with LSB Replacement embedding

LSBR, for practical reason, is the most commonly used technique for embedding. As illustrated in Figure. 3.5, this technique consists in replacing the least significant bits (LSBs) of the selected pixels with the bits of a message to be embedded (stego vectore). In other words, the least significant bit of each selected pixel is replaced by an element of the vector stego. However, it is important to note that it is possible to modify the pixels by replacing more than just one bit of the LSB, this is referred to as "2LSB replacement".

That being said, Steganography by LSB replacement is not very safe [Pevný et al., 2010b]. Indeed, even if the modifications made do not affect the external appearance of the medium, they considerably alter its statistical distribution leading to a phenomenon of probability equalization (stair-step effect in the histogram). As a consequence, a simple pairwise statistical analysis, such as $\mathcal{X}^2$, would be sufficient to detect the modification [Westfeld and Pfitzmann, 1999]. For very small

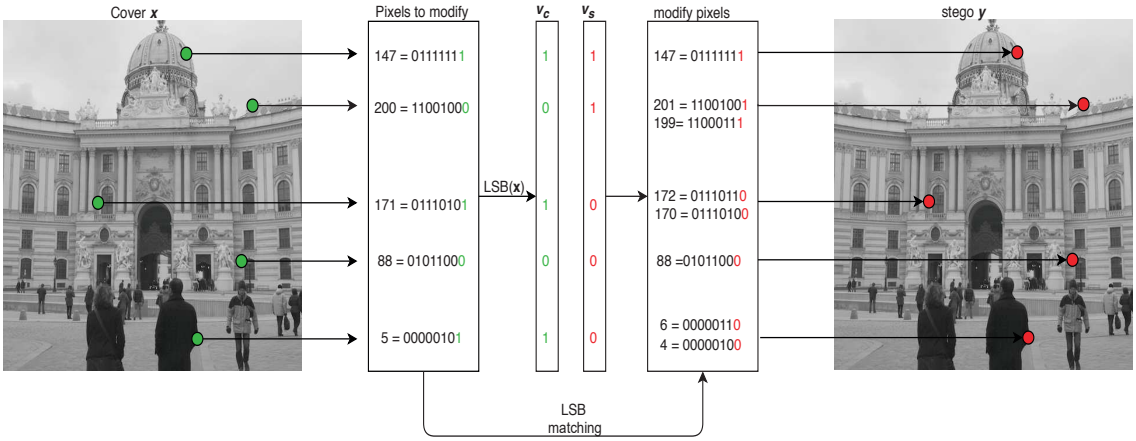



FIGURE 3.6: Illustration of the embedding algorithm in the case of an LSB matching. The cost map (3.3.1), as well as the key, will determine a sequence of pixels to be modified in the cover image $\boldsymbol{x}$. It is from these pixels that we generate the cover vector $\boldsymbol{v_c}$ through the extraction of the LSBs. A syndrome coding technique is applied to transform the cover vector into a stego vector $\boldsymbol{v_s}$ (explained in sub-section3.3.2.2). Finally, for each element of the stego vector $\boldsymbol{v_s}$ that differs from that of the cover vector $\boldsymbol{v_c}$, we have the choice between two bytes: the one obtained by adding 1, and the one obtained by subtracting 1. however, be careful with the values that are out of the interval $[0 : 255]$, otherwise it would be immediately detected by a steganalyst.

payload, machine learning [Fridrich and Kodovský, 2013] and deep learning [Chen et al., 2019] are very efficient.

### Steganography with LSB Matching embedding

In Figure. 3.6, we illustrate the embedding algorithm using the LSB matching, also known as $\pm 1$ embedding. This embedding algorithm is very close to the LSB replacement, as it also embeds the stego vector $v_s$ into the cover pixels. However, instead of replacing the LSB pixels, this approach works by randomly incrementing or decrementing the value of the pixel by 1. The LSB matching embedding method was proposed for the first time by [Sharp, 2001]. It was presented as a solution to overcome the LSB replacement low-security problem. Indeed, LSB matching techniques are much more difficult to detect compared to LSB replacement techniques. It is important to point out that all modern and secure algorithms are based on an embedding by "LSB matching".

In Section. 3.3, we first presented the concept of the adaptive steganography method, then reviewed the three essential elements of any adaptive steganography

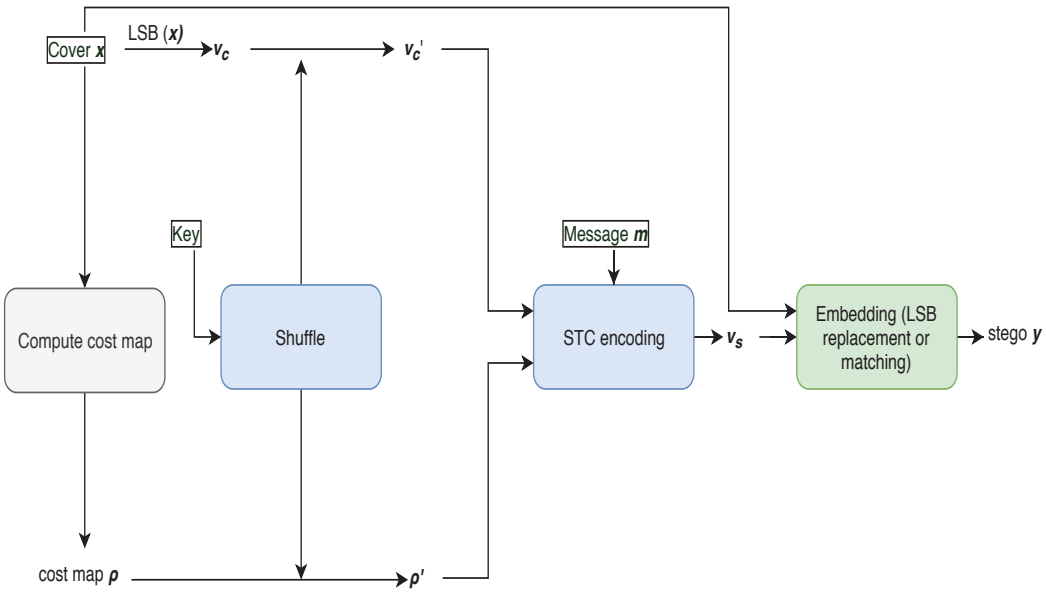



FIGURE 3.7: Embedding framework in adaptive steganography.

algorithm, from the computation of a cost map to the encoding of the message ending with the embedding method. In Figure. 3.7, we illustrate how to put these elements together to make an adaptive steganography algorithm.

### 3.3.4 Simulator

Using a specific coding scheme, such as STC (Section. 3.3.2.2, instead of optimal coding leads to a small sub-optimality in terms of embedding, this can be handled by using a simulator that simulates the embedding changes.

In [Fridrich and Filler, 2007] the minimal expected embedding distortion, for a *fixed payload* ( $m$), is given by the following equation:

$$minD(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} \rho_i \, p_i, \tag{3.11}$$

with $p_i$ the probability of modification of the $ith$ pixel, which is directly related to the detectability value $\rho_i$ of the pixel at position $i$. It is defined as follows [Pevný

FIGURE 3.8: Image from the BOSS base [Bas et al., 2011].

et al., 2010b]:

$$pi = \frac{e^{-\lambda\rho_i}}{1 + e^{-\lambda\rho_i}}, \tag{3.12}$$

with $\lambda > 0$ a constant determined by the constraint on the message's size:

$$-\sum_{i=0}^{n}(p_i \log_2 pi + (1 - p_i) \log_2(1 - pi)) = m, \tag{3.13}$$

The modification probability map can be then defined as follows: $\boldsymbol{p} = \{p_i, i \in [0, ..., n]\}$, where pixels with higher modification probability have a higher chance of being modified. The simulator carry out the embedding process by generating random values in the form of $\boldsymbol{r} \in [0, 1]_1^n$. These values are used later to decide how to change each pixel of the cover image $\boldsymbol{x}$ ($\pm 1$ or 0 modification).

We give in Figure. 3.9 the map of the modification probabilities associated with the image given in Figure. 3.8 when the HUGO [Pevný et al., 2010b] algorithm is used. Each pixel is associated with a modification probability.

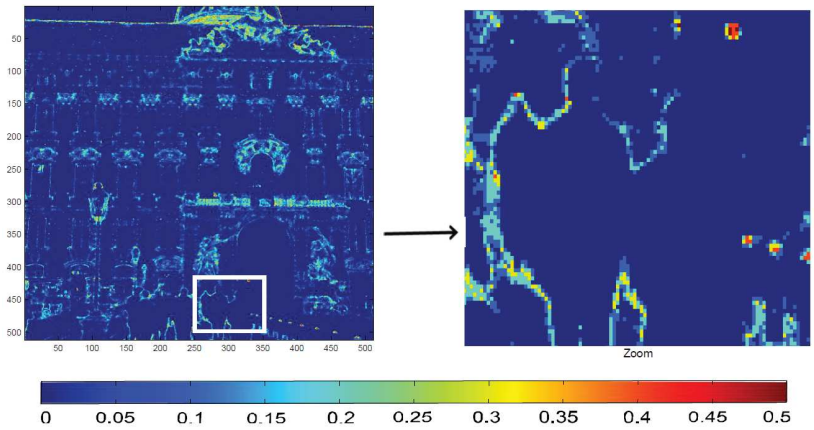



FIGURE 3.9: Embedding probability map of image in Figure. 3.8 when using Hugo at a payload of 0.2 bpp [Pevný et al., 2010b].

.

## 3.4 Embedding in spatial domain

Spatial domain embedding techniques carry out the embedding directly on the pixels. Thus, the embedding rate (the amount of data we embed for a given cover) is measured in bit per pixel (BPP). These techniques offer better embedding-capacity and less processing time compared to transform domain techniques where covers are transformed into another domain (i.e. frequency domain JPEG) before embedding.

In this section, we present some of the current state-of-the-art steganographic algorithms, highlighting how computing the cost map. The message encoding and embedding parts will not be discussed here as they remain identical to those illustrated in Figure. 3.7.

### 3.4.1 Highly Undetectable steGO (HUGO)

In this sub-section, we will present HUGO [Pevný et al., 2010b], the first adaptive algorithm proposed in the literature. It was proposed in 2010 and used in the same year as an embedding algorithm for the BOSS (Break Our Steganography System) competition [Bas et al., 2011].

The cost map $\boldsymbol{\rho} = \{\rho_i \in [0, \infty[\}_{i=0}^{n}$ of the HUGO algorithm is defined such that, for the *ith* pixel, the cost of its modification is:

$$\rho_i = \sum_{j=1}^{d} w[j] \, |\boldsymbol{f}_{\boldsymbol{x}}[j] - \boldsymbol{f}_{\boldsymbol{x} \sim \boldsymbol{x}_i}[j]|, \tag{3.14}$$

with:

- $\boldsymbol{f}$ is a feature vector generated from the co-occurrence matrix. Each *bin* from this matrix contains the number of occurrences of values triplets (d1, d2, d3) in the residual image. This latter is obtained by filtering the image with the kernel [1- 1], followed by a truncation in the interval $\{-T, ..., T\}$.
- $\boldsymbol{f}_{\boldsymbol{x}}$ refers to $d$-dimension feature vector of the image $\boldsymbol{x}$,
- $\boldsymbol{f}_{\boldsymbol{x} \sim \boldsymbol{x}_i}$ refers to the features vector of the image $\boldsymbol{x}$ whose *ith* pixel has been modified,
- $w[j]$ is the weight associated with the triplet $(d_1, d_2, d_3) \in \{-T, ..., T\}$, it is calculated as follow:

$$w[j] = \frac{1}{\left[\sqrt{d_1^2 + d_2^2 + d_3^2} + \sigma\right]^{\gamma}}, \tag{3.15}$$

with $\sigma$ and $\gamma$ are two parameters that can be tuned to minimize the detectability.

The term $w[j]$ is used to promote the embedding in images zones where the triplet $(d_1, d_2, d_3) \in \{-T, ...T\}$ is high, which tends to favour embedding in textured or contour zones. This can be noticed in Figure. 3.9.

Once the $\boldsymbol{\rho}$ cost map is found, we move on to the next step, where we select pixels to modify either by the help of a simulator, or by using STC. These pixels are then modified by $\mp 1$ depending on the distortion caused by each.

### 3.4.2 Spatial-UNIversal WAvelet Relative Distortion (S-UNIWARD)

S-UNIWARD is an embedding algorithm for the spatial domain. Its distortion function is defined in the wavelet domain. It made the first appearance through an article submitted to IHMMSec 2013 [Holub et al., 2014].

In S-UNIWARD the cost map $\boldsymbol{\rho}$ is composed of $\{\rho_i \in [0, \infty[\}_{i=1}^{n}$, where $\rho_i$ is given by the following equation:

$$\rho_i = \sum_{k=n}^{d} \sum_{u=1}^{n_1} \sum_{v=1}^{n_2} \frac{|W_{uv}^{(k)}(\boldsymbol{x}) - W_{uv}^{(k)}(\boldsymbol{x} \sim x_i)|}{\sigma + |W_{uv}^{(k)}(\boldsymbol{x})|}, \tag{3.16}$$

with

- $W_{uv}^{(k)}(\boldsymbol{x})$ the wavelet coefficient in the $(u, v) \in \{1, .., n_1\} \times \{1, ..., n_2\}$ position for the $kth$ sub-band of the image $\boldsymbol{x}$,
- $W_{uv}^{(k)}(\boldsymbol{x} \sim x_i)$ the wavelet coefficient in the $(u, v) \in \{1, .., n_1\} \times \{1, ..., n_2\}$ position for the $kth$ sub-band of the image $\boldsymbol{x}$ whose pixel $i$ has been modified,
- $\sigma$ a numerical stabilization constant [4].

Note that there are three filtering directions for wavelet decomposition: horizontal, vertical, diagonal. A closer look at the equation shows that the cost $\rho_i$ is small in textured areas. Indeed, for a pixel $i$, the intensity variations in the vertical, horizontal, and diagonal directions are obtained via wavelet decomposition. The higher the amplitude is in one or more directions, the larger the denominator is - and therefore the smaller the $\rho_i$; this indicates that it is possible preferably choose this pixel to make a modification.

[4] in [Holub et al., 2014] $\sigma$ is set to 1

### 3.4.3 Wavelet Obtained Weights (WOW)

The WOW algorithm [Holub and Fridrich, 2012], was proposed in 2012, this algorithm is very similar to S-UNIWARD considering the fact that the embedding costs are also calculated from three directional residues. It first calculates the weighted difference between the residual wavelet coefficients of the cover image, and the residual wavelet coefficients of the stego image and then aggregates the result obtained to construct a cost map. $\rho_i$ is given as follow:

$$\rho_i = \sum_{k=1}^{d} \frac{1}{\xi_i^{(k)}}, \tag{3.17}$$

with $\boldsymbol{\xi}^{(k)}$, known as embedding suitabilities, is computed as follow:

$$\xi_i^{(k)} = \sum_{u=1}^{n_1} \sum_{v=1}^{n_2} |W_{uv}^{(k)}(\boldsymbol{x})| \star |W_{uv}^{(k)}(\boldsymbol{x}) - W_{uv}^{(k)}(\boldsymbol{x} \sim x_i)|, \tag{3.18}$$

By taking a closer look at those equations, we notice that it tends to force the embedding modification to textured ones and avoid the smooth regions.

#### 3.4.3.1 A new cost function for spatial image steganography (HILL)

HILL is another embedding algorithm for the spatial domain, it was proposed in 2014 as an upgraded version of WOW. The authors [Li et al., 2014], when analyzing the cost map derived from Equation. 3.17 of WOW algorithm, have noticed the existence of some pixels with high-cost values inside texture regions; this is because the pixels may be predictable in one of the directions. However, a pixel in a texture region, even being predictable in one of the directions, should be assigned a lower cost value To this end, and to ensure that pixels within textured regions have relatively low costs, they replaced the three directional kernels with one non-directional high-pass filter which is then followed by two low-pass filters.

Based on the above analysis, they have proposed a cost map given by the following equation:

$$\begin{aligned} W^{(k)} &= \boldsymbol{x} \star H^{(k)}, \\ \boldsymbol{\xi}^{(k)} &= |W^{(k)}| \star L_1, \\ \boldsymbol{\rho} &= \sum_{k=1}^{d} \frac{1}{\boldsymbol{\xi}^{(k)}} \star L_2, \end{aligned} \tag{3.19}$$

with $W^{(k)}$ is the residual obtained by convolution the image $\boldsymbol{x}$ with a high pass filter $H^{(k)}$. $\boldsymbol{\xi}^{(k)}$ is the embedding suitability.

The high-pass filter is used to locate the less predictable parts in an image, where the two low-pass filters are used to make the low-cost values more clustered.

The given results (see next section) show that HILL algorithm makes the embedding changes more concentrated in texture region.

#### 3.4.3.2 Content-Adaptive Steganography by Minimizing Statistical Detectability (MiPOD)

MiPOD algorithm [Sedighi et al., 2016a], was proposed in 2016. It is one of the most secure embedding algorithms for spatial domain. This algorithm differs fundamentally from the previously presented algorithms as there are no pixel costs to start with. Instead, based on a residual model, the embedding change rates $\boldsymbol{p} = \{p_1, ..., p_n\}$ are first computed by solving numerically the two following equation:

$$p_i \sigma_i^{-4} = \frac{1}{2\lambda} ln \frac{1 - 2p_i}{p_i}, \qquad i = 1, , N, \tag{3.20}$$

$$R = \sum_{i=1}^{N} H(p_i), \tag{3.21}$$

with $\sigma_i^2$ is the variance of the cover's pixel $x_i$, that is computed using a variance estimator. $\lambda$ is the Lagrange multiplier.

Once the change rates are computed $\boldsymbol{p}$, using the method of Lagrange multipliers, they are converted to costs $\boldsymbol{\rho} = \{\rho_1, ..., \rho_n\}$ using the following equation:

$$\rho_i = \ln\left(1/p_i - 2\right) \tag{3.22}$$

These costs are then used by the syndrome-trellis codes to embed the payload $R$ during the embedding process.

#### 3.4.3.3 Discussion

Since 2010, steganography researchers have focused on developing well-designed cost functions in order to generate subtle cost maps that may lead to optimal adaptive embedding based on the content of the images. After 2016 and the MiPOD algorithm, there was no more major progress, except for some small improvement. Other steganography techniques have then emerged. We list below two techniques with strong potential.

**Natural steganography** In adaptive steganography, the steganographer tries to concentrate the embedding in certain regions of the cover, while considering the image content, for the objective of minimizing the distortion caused by the embedding.

However, even a small distortion will change the statistical cover distribution. Considering this fact, Natural steganography [Denemark et al., 2018] aims to embed the message by mimicking another cover source different from the initial one, i.e. a different ISO sensitivity. Thus the image obtained via the embedding is no more seen as stego but a cover from a different source. To do so, the added signal allows to switch from one source to another source.

**Strategic adaptive steganography** In adaptive steganography, the evolution of Eve's steganalysis strategy is not taken into account. It is then more interesting to propose an *optimal adaptive steganography* [Schöttle and Böhme, 2016], also called *strategic adaptive steganography*. With such a steganography algorithm, the pixels' modification probability is set to ensure the Nash equilibrium in the cat-and-mouse game between Alice/Bob and Eve (explained more in detail in next chapter.

## 3.5 Conclusion

In this chapter, we have introduced the modern notions of steganography, as well as the main methods for embedding. Among the presented embedding methods, we were particularly interested in adaptive steganography methods, which are based on the principle of minimizing a distortion function. For secret message embedding, we have seen that most current adaptive methods use a cost map, which assigns to each cover pixel a detectability cost $\rho_i$ (in the most of the case $\rho_i^+ = \rho_i^-$, reflecting its level of security during the embedding. Then we briefly presented the techniques of syndrome coding, we defined the concept of matrix embedding, and we presented some examples of corrective codes used for modern steganography, particularly STC. Then we have seen how to embed the stego vector within the medium carrier (cover image). Finally, we present a few recent embedding algorithms.

# Chapter 4

# Steganography using deep learning

Contents

As mentioned previously, Simmons [Simmons, 1984] has formalized the reasoning framework for the steganography/steganalysis domain as a *3 player game*. The steganographs, Alice and Bob, create a secret communication channel in order to converse secretly without being suspected by third-party (Eve). So, they use a banal medium, for example, an image, and dissimulate in this medium a message. Eve, the steganalyst, observes the exchanges between Alice and Bob. Its role is to make a binary decision, i.e. a two-class classification. In the case where the exchanges are images, Eve has to figure out whether they are natural image (cover images) or if they hide a message (stego images).

As discussed previously in Section. 3.3, modern embedding algorithms are adaptive which means that they take into account the content of the hosting medium (the cover) in order to better hide the message [Holub et al., 2014, Holub and Fridrich, 2012, Sedighi et al., 2016a]. Despite the fact that modern embedding approaches are the result of almost 20 years of research using codes and adaptivity, these algorithms, from a game theory point of view, are qualified as *naive adaptive steganography*. Indeed, when creating an embedding algorithm, the evolution of Eve's steganalysis strategy is not taken into account.

The game between Alice, Bob and Eve in Simmons model can be used in a game theory context. In this context, each player tries to find the strategy that maximizes his or her profits. For that purpose, we express the problem as a min-max problem that we strive to optimize. The optimum solution, if it exists, is called the Nash equilibrium solution. When all players use a strategy in the Nash equilibrium, the change in strategy of one player results in a counter-attack by the other players allowing them to increase their winnings. To this end, [Schöttle and Böhme, 2012, 2016] has proposed a formal solution for steganography named as *optimal adaptive steganography*, and also called *strategic adaptive steganography*. With such a steganographic algorithm, pixels, others than those modified by a naive approach, have a chance to be modified. In other words, in a *strategic adaptive steganography*, the pixels' modification probability is set to ensure the Nash equilibrium in the cat-and-mouse game between Alice/Bob and Eve and not only to minimize a distortion measure.

The *strategic adaptive steganography* is a very nice concept, but trying to formalize it mathematically often requires simplifying assumptions which are far from modelling the practical reality. Another way to obtain a Nash equilibrium is to "simulate" the game. Alice can play the game alone (from her side and without interacting with Bob or Eve) by using *three algorithms* which competes against each other (two algorithms in a simplified version). Those algorithms are the embedding algorithm, the extracting algorithm, and the steganalysis algorithm. They are named conventionally as *agents*; and more precisely *Agent-Alice* for the embedding algorithm, *Agent-Bob* for the extracting algorithm, and *Agent-Eve* for the steganalysis algorithm ( [Yedroudj et al., 2019]), thus making a distinction with the *Humans' users* Alice (sender), Bob (receiver), and Eve (Eavesdropper-in-the-middle).

As far as we know, the first attempt to simulate a *strategic equilibrium* dates back to 2011 with two approaches known as MOD in [Kodovsky et al., 2011] and ASO in [Kouider et al., 2013]. Whether for MOD or ASO, the game simulation is carried out through the rivalry between Agent-Alice and Agent-Eve. In both approaches, Agent-Bob is not used since Agent-Alice is simply generating a cost map, which is then used for coding and embedding the message thanks to an STC [Filler et al., 2010] (see Section. 3.3.2).

Both MOD and ASO are trained by repeating the two next steps until a stop criterion is reached:

- Agent-Alice updates the embedding cost map while requesting an Oracle how to update the embedding costs so that the corresponding stego is even less detectable (this is equivalent to an adversarial attack against a discriminant),
- The Oracle (Agent-Eve) updates the parameters of its classifier (SVM for MOD [Kodovsky et al., 2011] and an Ensemble Classifier for ASO [Kouider et al., 2013])

With the emergence of Generative Adversarial Networks (GAN) (discussed previously in Section. 2.3.1), many areas have been revolutionized; among them, there

is the strategic embedding. As a result, game simulation, already implemented in MOD [Kodovsky et al., 2011] and ASO [Kouider et al., 2013], becomes easier to deploy and to optimize through the use of neural networks. This has led to the emergence of new steganographic approaches based on the use of GANs and adversarial concepts. These approaches may be categorized into four families:

1. Approaches by synthesis/no modifications,
2. Approaches with probability map generation,
3. Approaches with adversarial embedding iterated,
4. Approaches with the 3 players concept.

## 4.1 Approach by synthesis with/without modifications

Approaches belonging to this family can be divided into two sub-categories.

1. approaches based image synthesis via a GAN with modification,
2. approaches based image synthesis via a GAN with no modification.

Approaches under the first sub-category proceed in two steps which are illustrated in Figure. 4.1. First, the GAN's generator G is trained to generate synthetic images which are then used as covers. The second step consists in embedding the secret message into a cover-generated image (by cover modification). The argument in favour of such approach is that the generated base would be more secure. One example of these approaches is the SSGAN [Shi et al., 2017], which was published on September 2017. Nevertheless, even if this protocol is feasible, it offers no advantage and only complicates the embedding process. It seems more reasonable to use an existing database from which Alice selects the image that best suits a secret message, with a lot of noise or textures [Sedighi et al., 2016b],

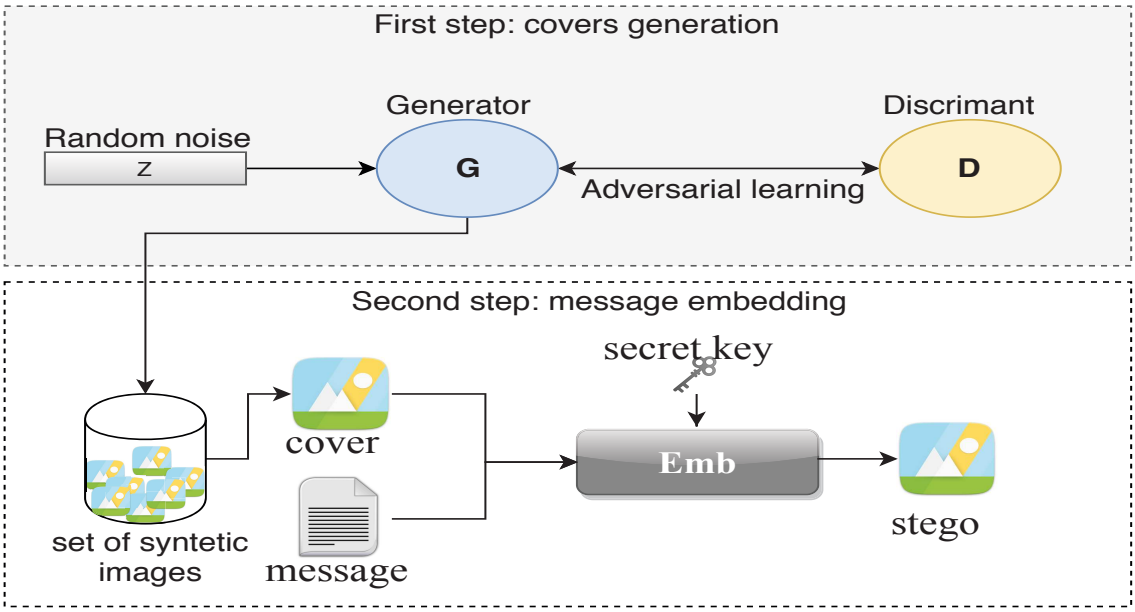



FIGURE 4.1: Workflow diagram of approaches by synthesis with modification.

poorly classified by a classifier [Kouider et al., 2012] or with a small deflection coefficient [Sedighi et al., 2016a].

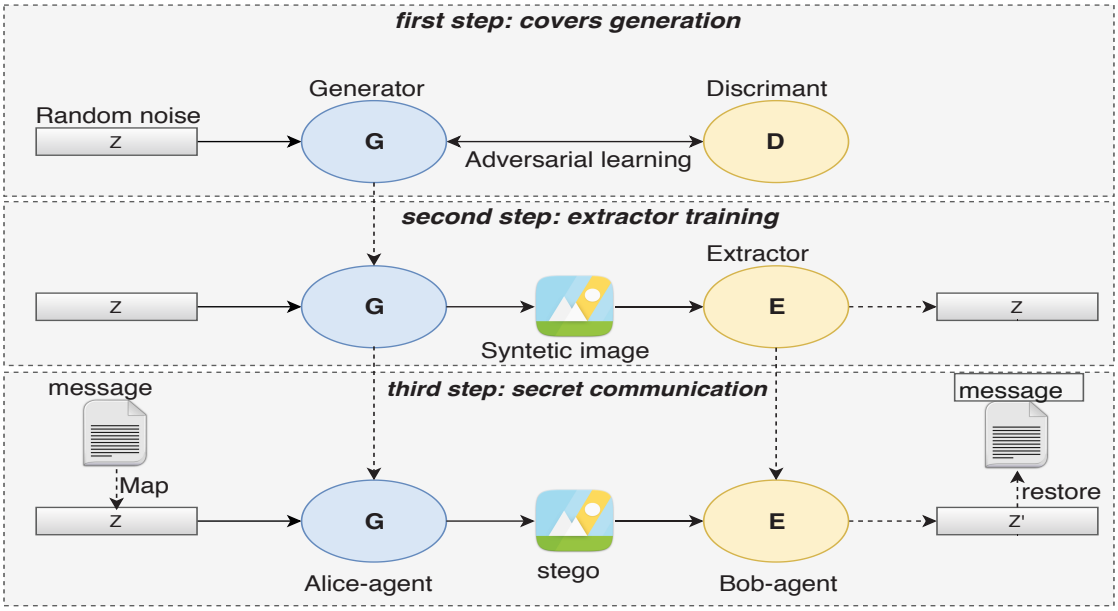



FIGURE 4.2: Workflow diagram of approaches by synthesis without modification.

The second sub-category includes approaches of image synthesis without modifications. This sub-category is much more interesting compared to the first one. This is because the stego generation is made directly by a neural network (the

generator of the GAN). Approaches in this category, as illustrated in Figure. 4.2, operate following the steps below:

1. pre-train a GAN to obtain generator G capable of generating synthesize images from a noise vector of fixed-size, and uniformly distributed in [-1; 1],
2. train another CNN, called the extractor E, to extract the noise vector from a synthesized image delivered by the generator G. The extracted vector must correspond to the one used by G when generating the synthetic image,
3. both the sender (Agent-Alice) and the extractor (Agent-Bob) hold the network and parameters of G and E, respectively. Thus, Alice, thanks to Agent-Alice network, can map the secret message to a fixed-size uniformly distributed vector, and then use it to generate a synthetic image, and then send it to Bob. Bob can extract the vector and retrieve the corresponding message through the use of Agent-Bob model.

As an example of this sub-category, we cite the following paper [Hu et al., 2018]. In this paper, the DCGAN generator [Radford et al., 2016] is used to synthesize images with a preliminary learning thanks to GAN methodology, and the size of the synthesized images is fixed to 64*64.

As already mentioned in a previous Section. 3.2.2, no modification approaches are considered as safe and secure approaches; however, one of the most significant drawbacks is the low number of bits that can be communicated compared to approaches with modifications.

## 4.2 Approach generating a probability map

In contrast to the first family, approaches from the "generation of a probability map" family are, exclusively, approaches with modifications. It means that a stego image is obtained by altering a giving cover image. The protocol of these approaches is resumed in Figure. 4.3.

The steganographic framework of these approaches is composed of two convolutional neural networks (a network for the generator G and the other for the discriminator D), and an embedding simulator. The role of the generator G is to generate, from a given image (cover), the modification probability map, which is then transformed to modification map whose values belong to [-1, 0, +1], thanks to the embedding simulator (an activation function). This modification map is then added (point-to-point sum) to the used cover image in order to obtain a stego image. The stego is then transferred with the cover to the discriminant network (steganalyst).

During the training phase, the discriminant's objective is defined as to minimize its detection error. As for the generator, it is set to create a modification map that reduces the detection accuracy of the discriminator (deceive the discriminator).

The family by generation of the modification probability map can be summarized in two papers: ASDL-GAN [Tang et al., 2017], and UT-6HPF-GAN [Yang et al., 2019]. We can also cite [Yang et al., 2019] which is another GAN that belongs to the "generation of a probability map" family but for JPEG images.

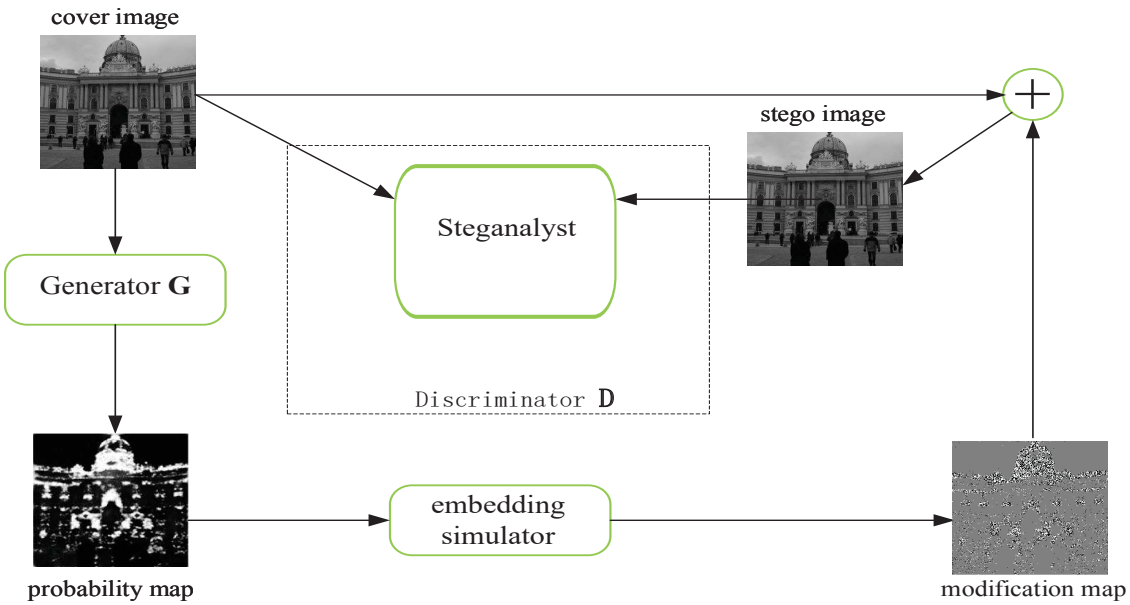



FIGURE 4.3: Workflow diagram of approaches with probability map generation.

For the ASDL-GAN, as illustrated in Figure. 4.4 (a), the discriminator D adopts the architecture of the Xu-Net model presented in Section. 5.5.1. In the generator

side, ASDL-GAN includes a convolutional neural network and an embedding simulator. The convolutional neural network is composed of 25 convolutional blocks. Each block is made of a succession of a convolutional layer, BN layer and an activation layer (ReLU excepts for the last block where a sigmoid is used). The generator network is used to generate, from a cover image, the modification probability map, which is then transmitted to the embedding simulator.

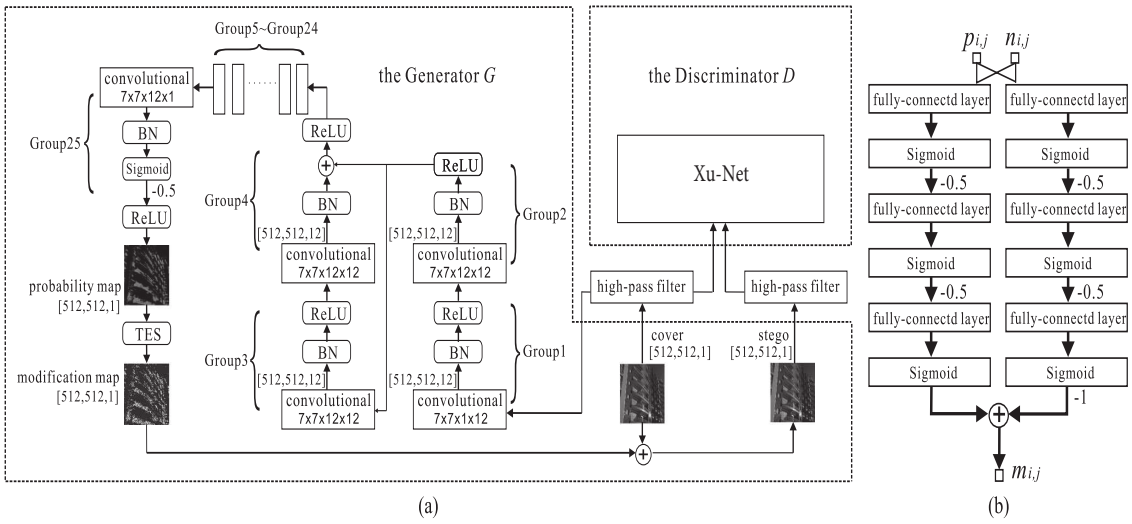



FIGURE 4.4: ASDL-GAN framework (extracted from [Tang et al., 2017]).

As for the embedding simulator, the authors have proposed the use of the TES (Ternary Embedding Simulator). As shown in 4.4 (b) TES is implemented as a mini neural network and composed of four parallel fully connected layers. This mini-network is first pre-trained to generate a modification map, from a modification probability map **p** and another map **n** of a random number drawn from a uniform distribution over the interval [0,1]. Once trained, the TES is used as an activation function by the generator to produce the modification map. The stego is then obtained through a point-to-point sum with the cover image.

Inspired by ASDL-GAN, UT-6HPF-GAN [Yang et al., 2019] is another framework for steganography within the generation of the modification probability map family. Employing the same component modules as ASDL-GAN, UT-6HPF-GAN includes a generator, an embedding simulator and a discriminator. For the embedding simulator, UT-6HPF-GAN employs an activation function (Tanh-simulator) instead of the TES in ASDL-GAN. This activation function is differentiable and

therefore allows the formal expression of the gradient and thus the propagation. This makes learning generator parameters easier and more efficient as compared to the ASDL-GAN. As for the generator, a more compact architecture based on U-NET [Ronneberger et al., 2015] is adopted. The discriminant remains the same as the one used on ASDL-GAN (Xu-Net [Xu et al., 2016a]).

## 4.3 Approach adversarial-embedding iterated

Another family of steganography is the adversarial embedding iterated. Approaches that fall within this family re-use the same game simulation concept discussed earlier in this chapter. However, instead of simulating the game using three agents (algorithms), Alice only uses two (Agent-Alice and Agent-Eve).

Historically, the idea of adversarial examples in steganography is not a recent one. However, the first truly adversarial iterated approaches are presented with MOD [Kodovsky et al., 2011] and ASO [Kouider et al., 2013]. Nevertheless, these approaches are not dynamic; they do not use a GAN.

In 2018, Tang proposed a new steganography scheme which is in the same spirit of the game simulation. This approach is named ADV-EMB [Tang et al., 2019] (previously named AMA on ArXiv arXiv:1803.09043). It re-uses the principle of ASO whose objective is to update the costs map while trying to mislead a steganalysis classifier (an oracle in ASO).

ADV-EMB algorithm operates by putting Agent-Eve (using the Xu-Net architecture [Xu et al., 2016a]) and Agent-Alice to compete against each other. Agent-Alice has access to the gradient of Agent-Eve loss and updates the costs map according to the gradients back propagated from the steganalyst (Agent-Eve). Therefore, the direction of modification of a cost value is more likely to be the inverse sign of the gradient. Before the game starts, the cost map values are initialized with the costs of S-UNIWARD.

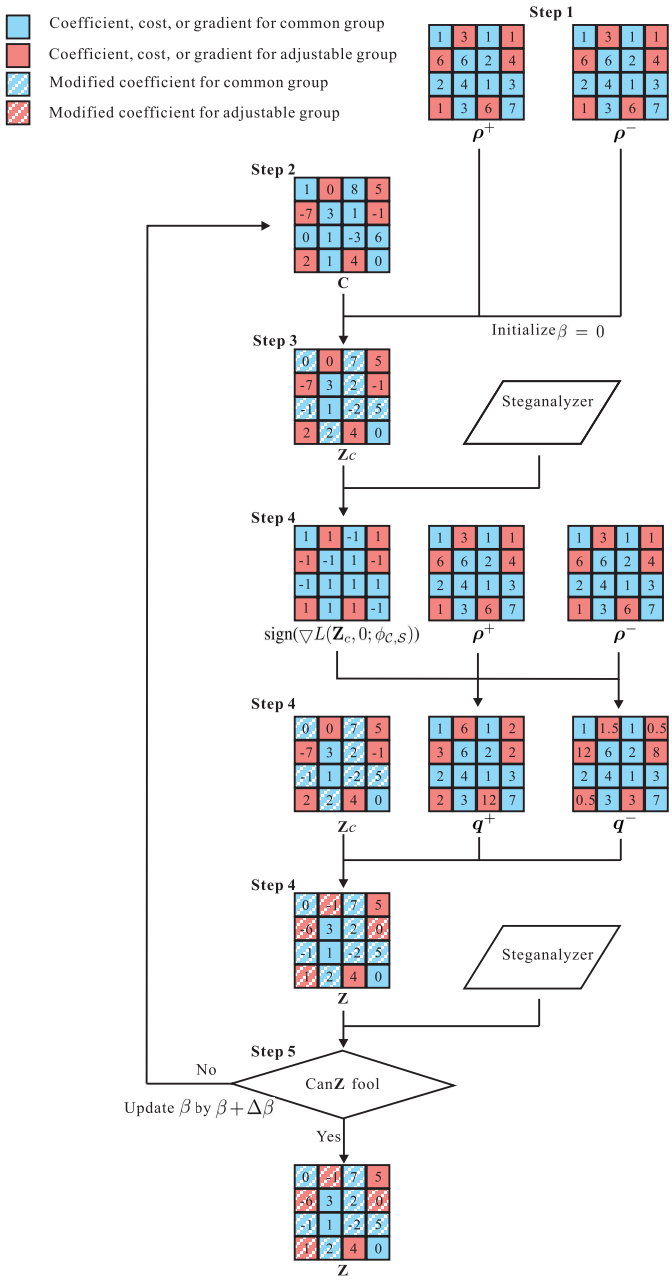



FIGURE 4.5: Illustration of the process of the proposed ADV-EMB scheme. extracted from [Tang et al., 2019].

The ADV-EMB scheme is illustrated in Figure. 4.5. In practical terms, ADV-EMB works by randomly splitting the image into two groups, i.e., a common group containing pixels used for modification-based embedding (illustrated with blue colour), and an adjustable group containing pixels to be modified to generate an adversarial sample that may deceive a steganalyst (red color in Figure. 4.5). The objective is to minimize the amount $\beta$ of adjustable pixels necessary to deceive a steganalyst (Agent-Eve). For that purpose, ADV-EMB repeats the following

steps, after initializing the value of $\beta$:

1. Embed a part of the secret message bits in the common group using an adaptive steganography algorithm (S-UNIWARD, presented in the Section. 3.4). The image $\mathbf{Z_c}$ is then obtained (with C is the cover image),
2. embed the remaining part of the message in the adjustable group, thus producing **Z** image. The adjustable pixels are modified so that Agent-Eve (the steganalyst) makes an erroneous prediction of the class label, based on the embedding costs for the adjustable elements q+, q-, whose their computation is explained in [Tang et al., 2019],
3. If the resulting image **Z** misleads the steganalyst successfully, then **Z** is considered as the final stego and therefore stop the iteration. Otherwise, increase the number of adjustable pixels $\beta = \beta + \Delta\beta$ and repeat the above steps.

The approaches within this family are achieving good results and attain a good security level. Moreover, even though the embedding algorithm is trained against a specific steganalyst, given the few traces that it introduces, it may not be easy to detect by another steganalyzer. This is probably because these approaches introduce only a limited number of modification to the initial cost map; thus, the initial embedding approach is likely to be preserved.

However, as it was the case for ASO, if the steganalyst (Agent-Eve) is not sufficiently effective to perform steganalysis, then this may have a totally counterproductive impact on Agent-Alice.

## 4.4 The 3 players approach

The last steganography GAN family is the 3-players game family. This family takes the concept of the previous family (Approach adversarial-embedding iterated in Section. 4.3) to the next level. So instead of two agents, 3-players game

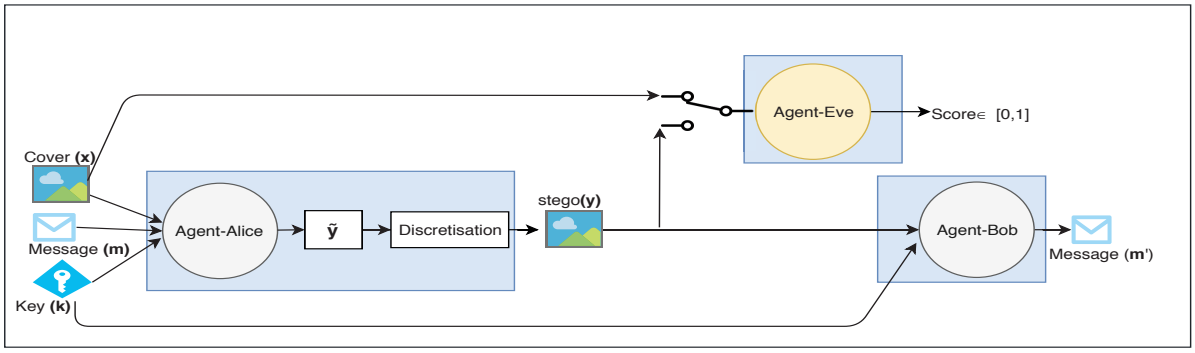



FIGURE 4.6: The overall architecture of the *3 player game*.

approaches use three agents: Agent-Alice (embedding algorithm), Agent-Bob (extraction algorithm), and Agent-Eve (the discriminant). It is important to mention that Agent-Alice and Agent-Bob are "linked" since Agent-Bob is only present to add a constraint to the solution obtained by Agent-Alice. Therefore, Agent-Alice and Agent-Eve play from their side an adversarial game where the objective is to learn an embedding algorithm. For the "game" between Agent-Alice and Agent-Bob, it is a cooperative game with the objective of making Agent-Bob capable of extracting the secret message without errors. We summarize the principle of the 3-player game family in Figure. 4.6.

The system takes as input a cover image **x**, a secret message**m** and a key**k**. These inputs are first introduced to Agent-Alice's network that generates a non-discrete-stego $\tilde{\mathbf{y}}$ ( $\tilde{\mathbf{y}} \in \mathbb{R}^{w \times h}$). Then the discretization module receives $\tilde{\mathbf{y}}$ and generates **y** a stego image with discrete values. **y** is then given to both Agent-Bob and Agent-Eve.

Agent-Bob tries to recover the secret message **m** from the stego **y** using the shared secret key **k**. Agent-Bob inputs (**y**, and **k**) go through a set of layers and mathematical operations; the extracted message **m'** is then generated.

Agent-Eve receives an image **z**, and returns the probability score of **z** belonging to the cover or stego class.

The training process of the *3 player game* is described in the following algorithm.

**Algorithm 1:** *3-player game* training process.

**Result:** *stegos, extracted_messages*
**Data:** *covers-list, messages-list, keys-list*
**while** *not converge OR loop* $\leq$ *max-iter* **do**
    `// Alice and Bob learning`
    **for** *iter_team*1 $\leq$ *it*1 **do**
        get_batch (*covers_list, messages_list,keys_list,batch_size*);
        forward-propagation (*covers, messages,keys*);
        update_Agent-Bob ($\mathcal{L}_{Bob}$);
        update_Agent-Alice ($\mathcal{L}_{Alice}$);
    **end**
    `// Eve learning`
    **for** *iter_team*2 $\leq$ *it*2 **do**
        get_batch (*covers_list, stegos_list*);
        forward-propagation (*covers, stegos*);
        update_Agent-Eve ($\mathcal{L}_{Eve}$);
    **end**
    *loop* + +;
**end**

As shown in **Algorithm 1** in line 1, the global system is trained at the maximum for *max-iter* "*loop*". In each loop, the learning is done sequentially by first, the team Agent-Bob and Agent-Alice (line 2) and then, the Agent-Eve (line 8). Note that there is a high number of loops in order to reach an equilibrium. Also, note that inside a loop there is also a certain number of back-propagation iterations for each agent.

Thus, for the learning of Agent-Bob and Agent-Alice (lines 2 to 7), there are *it1* iterations (line 2). For an iteration, we load a mini-batch of *cover images*, *secret messages*, and *keys* (line 3), we forward-propagate all the cover images on Agent-Bob and Agent-Alice networks (line 4), and then we update Agent-Bob and Agent-Alice by minimizing their losses, $\mathcal{L}_{Bob}$ and $\mathcal{L}_{Alice}$ respectively, thanks to the stochastic gradient descent (lines 5 and 6). During this learning, the weights of Agent-Eve are fixed.

The learning of Agent-Eve (lines 8 to 12) is similar to the learning of Agent-Bob and Agent-Alice. There are *it2* iterations (line 8). For an iteration, we load a mini-batch of *cover images* and *stego images* (line 9), we forward-propagate all the cover and stego images on Agent-Eve network (line 10), and then we update

Agent-Eve by minimizing Agent-Eve loss $\mathcal{L}_{Eve}$ thanks to the stochastic gradient descent (lines 11).

Since equilibrium is reached, the last agent playing the game will not change its strategy. Thus, the fact to be the last player, i.e. the last learning agent, will not impact the performances of the other agents.

To our best knowledge, the idea of three players has appeared for the first time in 2016, where Abadi and Andersen proposed a cryptographic toy example: an encryption algorithm using three Neural Networks [Abadi and Andersen, 2016]. The use of Neural Networks facilitates obtaining a *strategic equilibrium* since the problem is expressed as a min-max problem. Moreover, its optimization could be completed through the back-propagation optimization (discussed in Section. 2.2.5). Naturally, this *3 player game* concept can be transposed in the steganography domain using deep learning.

In December 2017 [Hayes and Danezis, 2017], and September 2018 [Zhu et al., 2018], two different teams from the machine learning community proposed, in NIPS 2017 and ECCV 2018, two steganographic systems based on a 3 player game approach. Both use 3 CNNs iteratively updated. These CNN play the roles of the Agent-Alice, Agent-Bob, and Agent-Eve. Those two papers fly over the *3 player game* concept, but their assertions are faulty, mainly because the security notions and their evaluation are not treated correctly [Yedroudj et al., 2019]. This leads to a 'Non-functioning' system for an embedding purpose. When Eve is clairvoyant, these two approaches are, in reality, very detectable.

## 4.5 Conclusion

In this chapter, we reviewed image steganography with GAN. We first give the principle of strategic adaptive steganography, pointing the difficulties of adopting the game theory concepts on steganography fields. This chapter focuses mainly on the GAN-based steganography methods, which are categorized into four families.

First is the synthesis family. This family is also divided into two sub-categories: with and without modification. For the modification methods, the GAN is used to construct the original carrier. As for the no-modification methods the GAN is responsible for the generation of the stego image. Second, we have a family of generation of the modification probability map. Methods that belong to this family use GAN to generate a modification map which is summed up with the cover image to obtain the stego image. Approaches with adversarial embedding iterated form the third steganography by deep learning family. The idea behind these approaches is to put the generator and the discriminant in competition to learn an embedding algorithm. The last family is the 3 player game. As its name suggests, approaches from this family use three agents where a neural network represents each agent.

# Chapter 5

# Steganalysis in spatial domain

## Contents

## 5.1 General presentation

Steganography's objective is to hide the presence of a secret message and to create a covert channel. For that purpose, steganography by cover modification algorithms (presented previously in Section. 3.2.1) operates by embedding the secret message within a digital media (*cover*). To assure a good security level, the embedding process is guided by a distortion function. This function is used to minimize the impact induced by the embedding process, thus, making the algorithm more secure. However, none of the current steganographic algorithms attains perfect security [Cox et al., 2007]. This is because steganographic algorithms introduce small changes (artefacts) during the embedding process. These changes may be leveraged by an attacker to identify altered images (stegos) from covers. The attacker is known as *steganalyst*, while the techniques used by the steganalyst are called steganalysis.

Steganalysis is the counterpart of steganography. It is defined as the art or science of detecting hidden data in suspicious files. Note that steganalysis does not have as its initial objective to extract the data hidden using a steganographic algorithm, it consists only of detecting the presence of hidden data. In other words, the main purpose of steganalysis is to identify whether or not a given medium is used as a steganographic carrier. Jessica Fridrich in [Fridrich et al., 2002] stated that the ability to detect a hidden message in a given image (stego) is related to the length of the message embedded in that image. Indeed, the smaller the size of the secret message, the less cover-change is carried out (during the embedding process), and therefore the slighter the possibility of introducing detectable artefacts.

## 5.2 Different classes of steganalysis

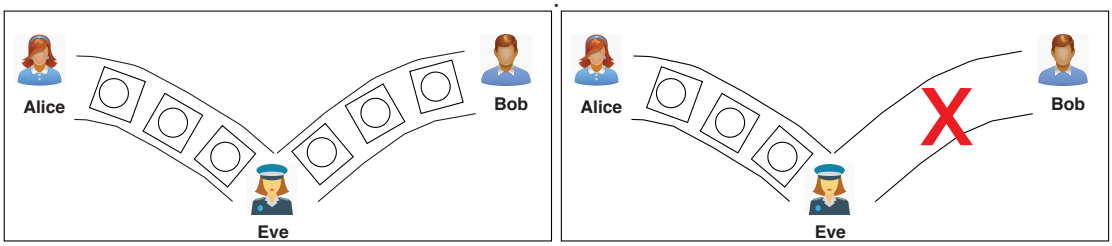
In the field of steganalysis, three main classes can be distinguished (active, passive, malicious steganalysis). These steganalysis classes are identified according to the action that the warden Eve, from the Simmons' model, takes when she detects

a secret communication. Mostly, Eve is supposed to be a passive warden, which means she has no right to interfere with Alice and Bob's communication. We discuss by following this option among other options of the warden Eve:

- The passive warden scenario: Eve in this scenario shall not interfere with the content of the communication channel, as its goal is only to detect the mere presence of secret communication. The passive warden can only prevent or authorize the delivery of the message; no message sent can be destroyed nor modified.

  As indicated in Figure. 5.1, when Alice sends a message to Bob (or the other way around), the message goes through Eve who analyses this message. If the message is considered as "clear", (see Figure. 5.1 (a)) then it will be delivered to the concerned person; otherwise, the communication channel is cut off [Anderson and Petitcolas, 1998], see Figure. 5.1 (b).

  Nevertheless, if Bob does not receive the message he is waiting for, he may suspect that a steganalyst has managed to cut off the communication, in such case Alice and Bob will modify the used steganography techniques and send the message back.




a) Eve passive warden when secret message is not detected

b) Eve passive warden when secret message is detected

FIGURE 5.1: Eve is passive warden: In the case of no secret message is detected, the sent file can be further transmitted through the communication channel, as shown in (a). However, if a secret message is detected, the warden will block the transmission and Bob will not receive the file (b).

- The active warden scenario: As shown in Figure. 5.2, the objective of the active warden is to detect, alter and eventually delete a secret message hidden within the sent message (cover). The altered cover is then re-sent to the other prisoner, and thus, he (or she) will not suspect that their system was

compromised [Ettinger, 1998, Cachin, 1998]. In an active warden scenario, Eve wants to destroy the secret message by applying some modification in the received object. For example, compression can be applied by an active warden to modify a digital image sent from Alice to Bob.

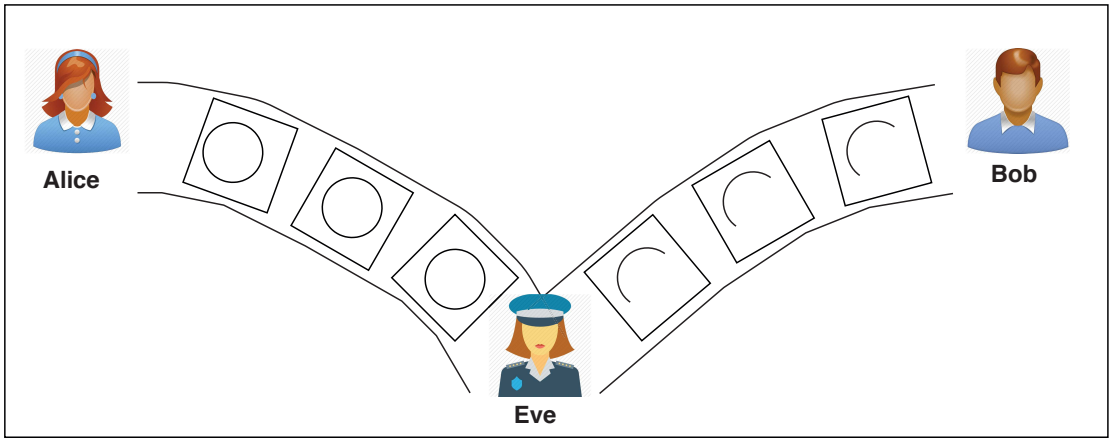



FIGURE 5.2: Eve is active warden: in this case when a secret message is detected Eve objective is to alter the intercepted medium sufficiently to preserve only the perceptible content and prevent the extracting and reading of a possible hidden message.

- The malicious warden scenario: In this case, and as illustrated in Figure. 5.3, when the warden detects the presence of secret message, she will attempt to understand the used steganographic system in order to modify the sent message or even produce new messages to impersonate the sender and mislead the receiver [III and Gomez, 2006].

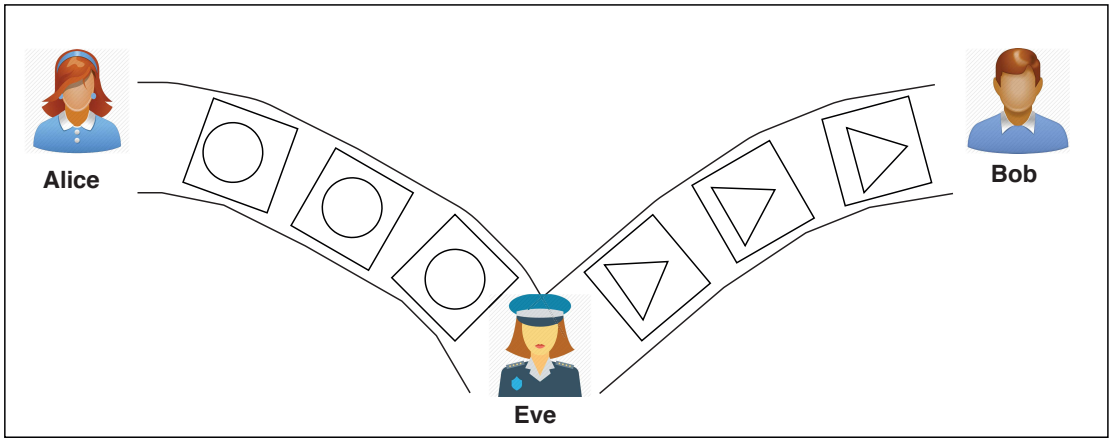



FIGURE 5.3: Eve is malicious warden: when a secret message is detected, Eve will try to understand the used steganographic technique so she can reintroduce a falsified message.

Note that this scenario is difficult in practice because it is necessary to maintain semantic consistency in communication and also to be technically capable of doing so. [Craver, 1998] The works presented in this manuscript consider only the first class of passive steganalysis.

### 5.2.1 Specific versus generic steganalysis

In general, there are two categories of steganalysis that are summarized in Figure. 5.4. 1) targeted steganalysis, which is more a forensic steganalysis approach, 2) blind steganalysis, which is a generic approach.

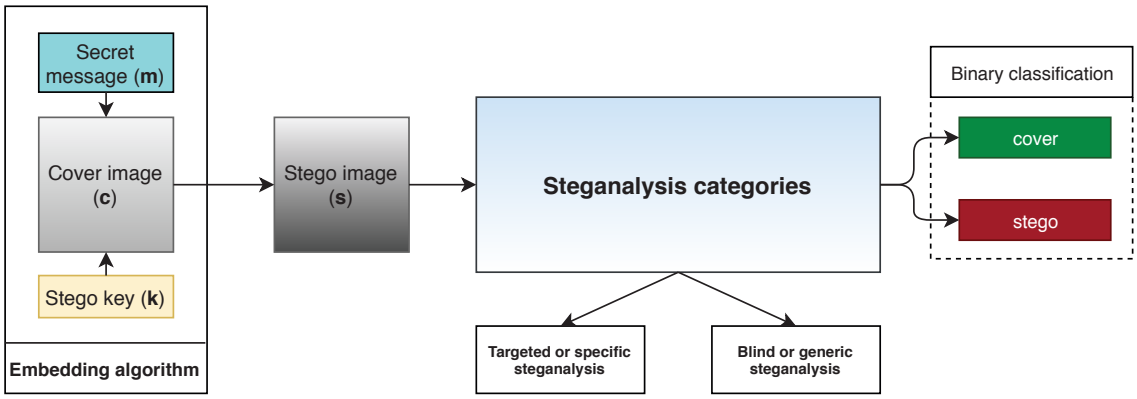



FIGURE 5.4: Specific and generic steganalysis.

**Specific steganalysis**, also known as targeted steganalysis, is based on the practice of attacking a specific steganographic algorithm through the identification of its security weaknesses (statistical flaws). The steganalyst analyzes the implementation and details of a specific embedding algorithm to find a weakness that can lead to statistical traces in the stego. In other words, targeted steganalysis is based on the identification of implementation flaws and weakness that introduce specific patterns that may characterize a steganographic algorithm. This type of attack is effective when the steganalyst is tested against stego images generated by a known embedding algorithm. However, the steganalyst can fail considerably if tested against stego images resulting from an algorithm other than the one being attacked.

As an example of targeted steganalysis, we have the attack developed by [Kodovsky et al., 2011]. This targeted steganalysis method is dedicated to the detection of any HUGO algorithm [Pevný et al., 2010b]. In this attack, a 4-d feature vector (see Section. 5.4.1) was sufficient to detect HUGO algorithm.

Nowadays, in the era of adaptive steganography algorithms, it is almost impossible to find an embedding algorithm with exploitable weakness, hence targeted steganography is not an "up to date" paradigm.

**Blind steganalysis**, also named generic steganalysis, is the current paradigm used to attack a steganographic scheme. Unlike targeted steganalysis, blind steganalysis techniques are not designed to attack one specific embedding algorithm which is a more general approach which it is supposed to detect different types of steganography content.

Because blind steganalysis can detect a broader class of steganographic techniques, it is generally less accurate; however, blind steganalysis is an irreplaceable detection tool if the embedding algorithm is unknown or secret.

Blind steganalysis is mainly based on the use of machine learning methods (discussed in the previous chapter 2). The objective of this type of approach is to learn the features that characterize a cover image and those that characterize stego image, in order to distinguish between the two classes of cover and stego. In practice, this is carried out either by using machine learning techniques seen in Section. 2.1 or by using deep learning techniques. In the first case, the steganalyst starts by extracting relevant features that allow separating the two classes (cover and stego), then a particular classifier is used to learn a separation function of the two classes cover/stego. In the second case, the features extraction and classification are regrouped together in one block (see Section. 2.2). Three examples of blind steganalysis are presented in [Fridrich, 2004, Shi et al., 2006, Xu et al., 2016a]. In this thesis, we interest only in blind steganalysis.

## 5.3 Steganalysis scenario

In steganalysis, there are several possible scenarios. These scenarios define several rules and assumptions on what the steganalyst knows about the steganographer's embedding process. In this section, we present some of the different scenarios.

### 5.3.1 Clairvoyant scenario

Kerckhoffs declares with the so-called Kerckhoffs principle Kerckhoffs [pp. 5-38 Jan. 1883, pp. 161-191, Feb. 1883] that the security of the message must not depend on the secret of the algorithm (which must be publicly released), but only on the security of a secret key.

Clairvoyant steganalysis operates within the context of this principle, as the security of the steganographic scheme relies only on the secret key. In this scenario, the warden Eve has a rough idea of **probability distribution of the cover images**. In addition, except for the secret key, Eve has all the elements of the steganographic scheme (**payload**, and **embedding algorithm**,**image size**), i.e. she can roughly guess the distribution of stego images. Thus, the role of the steganalyst Eve is reduced to examining whether an intercepted image is a cover image or stego image.

Formally, for a given image $\boldsymbol{i} = (i_1, ..., i_n)$, the secret message detection problem can be represented as a test between two hypotheses:

$$\begin{cases} H_0 : \boldsymbol{i} \sim \boldsymbol{P}_c & \text{the image } \boldsymbol{i} \text{ is a cover image} \\ H_1 : \boldsymbol{i} \sim \boldsymbol{P}_s & \text{the image } \boldsymbol{i} \text{ is a stego image} \end{cases} \tag{5.1}$$

with $\boldsymbol{P}_c$ the distribution of the cover images, and $\boldsymbol{P}_s$ the distribution of stego images.

From a steganographer point of view, the clairvoyant steganalysis is the most difficult scenario. Indeed, for the steganalyst, it is easier to develop an effective attack knowing almost all parameters. In recent years, a lot of work has been done on this scenario, both in steganography [Kodovsky et al., 2011, Holub and Fridrich, 2012] or in steganalysis [Fridrich et al., 2011, Pibre et al., 2016, Kodovský et al., 2012, Fridrich and Kodovský, 2012].

### 5.3.2 Clairvoyant scenario with side channel informed

The development of increasingly sophisticated adaptive steganography (Section. 3.3) has given considerable attention to the design of adaptive steganalysis schemes, **clairvoyant steganalysis with Selection Channel Aware** (SCA).

This scenario can be considered as an upgrade of the clairvoyant steganalysis scenario. In the context of this scenario, Eve, the steganalyst has access to steganographic scheme but also, to an additional information related to content-adaptive steganographic algorithms, which is **the modification probability map** (the selection channel).

It is assumed that Eve knows or can have a good estimate of the modification probability map since she knows the embedding scheme and can compute the embedding costs and deduce the probability map, or she can simply embed multiple times in an image and directly estimate the modification probability map.

Incorporating this extra information (the modification probability map) into an already existing steganalysis scheme can improve its performance, especially for a low embedded payload [Tang et al., 2014]. For this scenario, there are two possible practical approaches:

1. The handcrafted approach based on the use of machine learning techniques. Among the most well-known methods that belong to this approach, we find MaxSRM [Denemark et al., 2014], tSRM [Tang et al., 2014], $\sigma$maxSRM [Denemark et al., 2016b].

2. The modern approach based on the use of deep learning techniques such as SCA-Ye-Net and SCA-SRNet, which are the selection channel aware versions of Ye-Net [Ye et al., 2017] and SRNet [Boroumand et al., 2019], respectively.

### 5.3.3 Mismatch scenarios

Clairvoyant steganalysis (with and without selection channel) is currently the methodology that achieves the best performances, this is explained by the fact that the steganalyst is positioned in a controlled environment, where both the steganographic scheme, the payload size and also the image source are known. However, in the case where one of these three parameters becomes unknown, the performance of the steganalysis scheme may be reduced. This scenario is called mismatch steganalysis. We can distinguish two types of mismatch, cover-source mismatch (different sources: different cameras, or different development such as interpolation, demosaicing, and gamma correction ...etc.) [Cancelli et al., 2008] and stego mismatch (different stego).

The problem of Cover-Source Mismatch (CSM) occurs when the images to be steganalyzed come from unknown sources [Fridrich et al., 2011]. Indeed, in practice, the images used during the steganalyst training do not come from the same source as the images used during the tests. In such a case, it is essential to determine what characterizes a cover distribution to avoid the strong dependence on the cover distribution during training. This kind of mismatch was identified, as far as we can tell, during the BOSS contest. The steganalyst was first trained on the BOSSBase [Bas et al., 2011], once trained, it was tested on real-world images that did not belong to the training set. The conclusion was that performance dropped down dramatically compared to the tests performed on the BOSSbase image.

One possible solution to avoid the CSM is to used a huge dataset to ensure image diversity and thus reduce the chance of encountering a cover image of an unknown distribution [Lubenko and Ker, 2012a], [Lubenko and Ker, 2012b]. This approach is referred to as "holistic approach". The other way is "the atomistic approach,"

i.e. to pre-process the image database in order to partition it into a few clusters where images that share similar features are grouped together. Then, a particular classifier is assigned to each cluster to learn how to classify its vectors [Pasquet et al., 2014].

The Stego Mismatch (SM), is similar to the CSM scenario. However, in this scenario, the parameter considered as unknown for the steganalyst is no more the cover source (probability distribution of the cover images) but the steganographic scheme, i.e. the probability distribution of the stego images. In practice, the embedding algorithm used to generate the stego images used during the steganalyst's training are not necessarily those present in the test-set.

Another kind of Stego Mismatch can be triggered when the steganalyst has no access to the payload size information. Despite the fact that the steganalyst knows the steganographic scheme, she/he is unable, without knowing the payload size, to access to the probability distribution of the stego images. Thus training and testing the steganalyst on two different distribution will lead to a mismatch problem.

In this thesis, we will only consider the clairvoyant scenario.

## 5.4 Steganalysis using two-step learning approaches

As mentioned before, steganalysis is considered as a hypothesis-testing problem where a steganalyst needs to determine whether or not, a given digital medium is "stego". Therefore, steganalysis can be treated as a binary classification problem (two classes). This implies that steganalysis may be addressed using machine learning tools (ML) (see Figure. 5.5) and more recently, deep learning tools. ML-based steganalysis generally follows the following classification frameworks.

Let $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s)}, y^{(s)})\}$ a dataset with $s$ sample. $\boldsymbol{x}^{(i)} = (x_1, ..., x_n) \in \{0, ...255\}^n$ is a 8-bit grayscale images, $y^{(i)} \in \{0, 1\}$ is the corresponding label, $y = 0$ if $\boldsymbol{x}^{(i)}$ is a cover, and $y = 1$ if $\boldsymbol{x}^{(i)}$ is a stego. We want thus to learn a

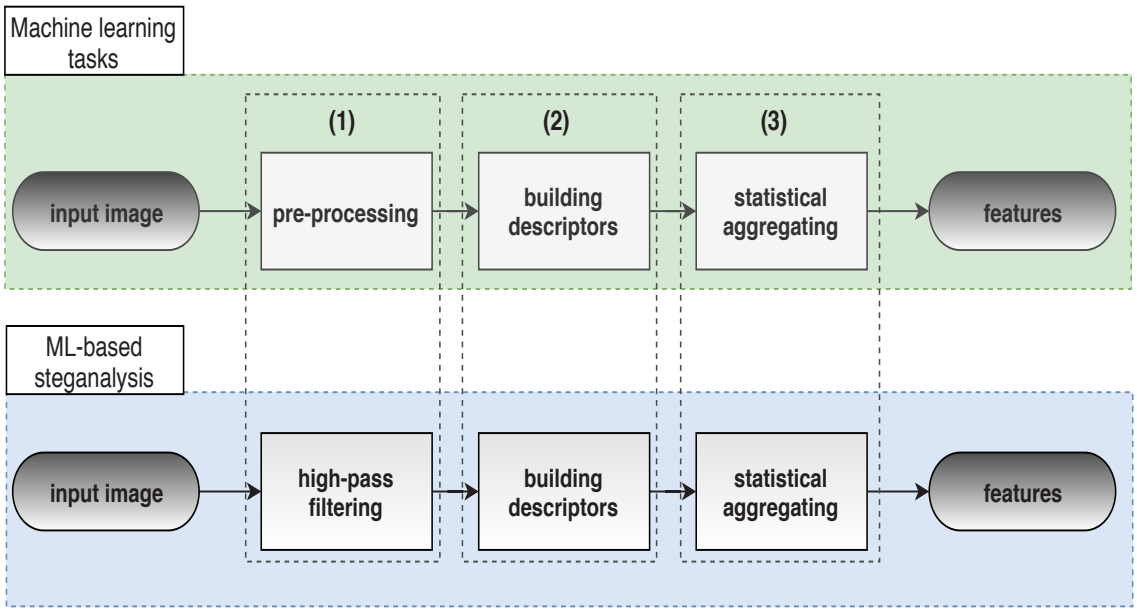



FIGURE 5.5: Steganalysis with regard to machine learning tasks.

mapping function $f : x \rightarrow y$ that can give the most accurate prediction $\hat{y}^{(i)}$ of the image $\boldsymbol{x}^{(i)}$ compared to its correct label $y^{(i)}$ where:

$$\hat{y}^{(i)} = f(\boldsymbol{x}^{(i)}) \tag{5.2}$$

In practice the mapping function $f(.)$ is learned through the minimization of the probability of error given in Equation. 5.3, over a part of the dataset $\mathcal{D}$. This part of the dataset is known as training set $\mathcal{D}_{train} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s_{train})}, y^{(s_{train})})\}$. Once learned, the accuracy of $f(.)$ is empirically evaluated on the other part of the dataset, called testing set and refereed as $\mathcal{D}_{test} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s_{test})}, y^{(s_{test})})\}$, satisfying $s = s_{train} + s_{test}$ and $s_{train} \cap s_{test} = \varnothing$

$$P_e = \frac{1}{s_{train}} \sum_{i}^{s_{train}} |y^{(i)} - \hat{y}^{(i)}| \tag{5.3}$$

ML-based steganalysis, like any other machine learning application, involves two steps: 1) features extraction, 2) classification with learnable classifier.

The feature extraction step consists in applying the hand-crafted transformation function $f_{ext}(.)$ to transform the database $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(s)}, y^{(s)})\}$ to $\mathcal{D}' = \{(\boldsymbol{z}^{(1)}, y^{(1)}), ..., (\boldsymbol{z}^{(s)}, y^{(s)})\}$, with $\boldsymbol{z}^{(i)} \in \mathbb{R}^m$ is $m$-dimension feature vector.

The obtained database $\mathcal{D}'$, more precisely $\mathcal{D}'_{train}$ is used to train a generic classifier to learn a mapping function $f_{classif}(.)$, which maps feature vectors $\boldsymbol{z}^{(i)}$ to a label $y^{(i)}$ (stego, cover). Therefore, the mapping function $f(.)$ is a composition of the two function $f_{ext}(.)$ and $f_{class}(.)$.

$$\hat{y}^{(i)} = f(\boldsymbol{x}^{(i)}) = f_{classif}(f_{ext}(\boldsymbol{x}^{(i)})), \tag{5.4}$$

Despite the importance of the learnable classifier and its direct impact on the performance of the machine-learning-based methods, features extraction is generally the most crucial step. A well-designed and sophisticated feature extraction allows not only to reduce the search space of the classifier chosen by the steganalyst, but also a good separation of the required classes (in our case the cover and stego classes). Therefore, it is important to choose features that best discriminate cover images from stego images. In this thesis, ML-based steganalysis methods are refereed as two-step learning approaches.

In common with other machine learning tasks, feature extraction in steganalysis based two-step learning approaches generally consists of three essential steps (see Figure. 5.5):

1. computing 'residuals' (the residual noise images);
2. building descriptors that reflect the links between each pixel and its neighbours over each residual;
3. producing a feature vector by aggregating the descriptors together.

In this context, several studies have been proposed that follow the same typical 3-steps procedure such us SPAM [Pevný et al., 2010a], SRM [Fridrich and Kodovský, 2012], PSRM [Holub et al., 2013], CCJRM [Kodovský and Fridrich, 2012].

For the classification step, a classifier is used to learn from the feature vector delivered by the feature extraction algorithm. The most commonly used classifiers are SVM [Hearst, 1998], and Ensemble classifiers [Kodovský et al., 2012].

### 5.4.1 Feature vector extraction (Rich Model)

Unlike most of pattern recognition tasks, in steganalysis, the signals of interest are the embedding modification hidden in the cover image pixels. Thus, the extracted features must be sensitive to steganographic modifications and insensitive to the content of the image. To this end, Fridrich have proposed a feature extractor that follows the typical 3-steps features extraction, and in the same time is well adapted to steganalysis particularity known as Rich Model (RM) [Fridrich and Kodovský, 2012]. In the following, we will present the functioning of this feature extractor (RM).

#### 5.4.1.1 Computing residuals:

As pointed before, steganalysis is a classification problem with extremely low signal-noise-ratio (SNR). To boost the SNR, RM algorithm performs a high-pass filtering in early stages of the feature extraction process in order to suppress the irrelevant image-content signal, and to extract and model the residual noise noted $\boldsymbol{r} = (r_{ij}) \in \mathbb{R}^{n_1 \times n_2}$ which is computed using the high-pass filters of the following form:

$$r_{i,j} = \hat{X}_{i,j}(\mathcal{N}_{i,j}) - cx_{i,j} \tag{5.5}$$

with $\mathcal{N}_{i,j}$ the neighborhood of the pixel $x_{i,j}$, $(x_{ij} \notin \mathcal{N}_{i,j})$, $c \in \mathbb{N}$ the order of residual noise, and $\hat{X}_{i,j}$ a prediction function that estimates the value of the cover pixel $x_{i,j}$ from its neighborhood $\mathcal{N}_{i,j}$.

The residuals generated from the above equation are called SPAM residuals. To introduce a certain non-linearity into the calculation of the residual, SPAM residuals are processed to generate MINMAX residuals by element-wise taking the minimum or maximum of the corresponding pixel values in multiple residuals.

Once calculated, the residual noise is then quantified and truncated according to the following equation:

$$r_{i,j} = Trunc(round\left(\frac{r_{i,j}}{q}\right), \tag{5.6}$$

where q > 0 is a quantification step, and $Trunc()$ is a truncation function that is defined as:

$$Trunc\,(x) = \begin{cases} T & \text{if } x \geq T \\ -T & \text{if } x \leq -T \\ x & \text{otherwise ,} \end{cases} \tag{5.7}$$

with $T \in \mathbb{N}$ is a threshold parameter.

Quantification is an essential step, as it allows the analysis of different ranges of noise amplitude (low amplitude: flat, medium and high amplitude: borders and textured areas). Truncation, on the other hand, reduces the range of residual noise values (by taking a small T). Both the truncation threshold T and quantization q are empirically determined to be T = 2 and $q \in \{1, 2, 3\}$.

In essence, the combined effect of quantification and truncation is equivalent to the reduction of the dynamic range of residues; this is to facilitate the production of co-occurrence matrices sensitive to the embedding modification changes (see next section on co-occurrence matrices) for accurate statistical modelling.

#### 5.4.1.2 Building descriptors (co-occurrence matrices):

The second step for the feature extraction is descriptors building. In RM this consists of the calculation of the co-occurrence matrices on the neighbourhood of

the residuals elements. During this step, the steganalyst calculates, the probabilities of occurrence in the residuals matrices resulting from the first step (after quantification and truncation).

In practice, RM operates by assigning to each element in $\boldsymbol{r}$ , denoted as $r_{i,j}$ a descriptor value. This value is computed based on the values of the four consecutive neighbors of $r_{i,j}$ over the residual matrix in **h**orizontal, **v**ertical directions, or **d**iagonals directions e.g., for a vertical direction, the co-occurrence value of a 4 neighborhoods-pixel is calculated by:

$$C^{v}_{(k_0,k_1,k_2,k_3)} = \sum_{i,j} [(r_{i,j}, r_{i+1,j}, r_{i+2,j}, r_{i+3,j}) = (k_0, k_1, k_2, k_3)] \tag{5.8}$$

with [.] kronecker operator, and $\{k_0, k_1, k_2, k_3\} \in [-T, ..., T]$.

It is important to enlighten the reader on the fact that the spatial information is discarded when constructing the co-occurrence matrices. This makes the classifier less sensitive to features spatial location and thus prevents from reducing the steganalysis problem to the memorization of the embedding locations. Also, it makes the process less sensitive to the size of test-set images.

#### 5.4.1.3 Producing feature vector

This step is intended to construct the final feature vector, $\boldsymbol{z} \in \mathbb{R}^m$, of high dimension by merging the different features from the different co-occurrence matrices and different filters.

In order to reduce the dimensionality and generate more compacted and robust features, both the symmetric natures of co-occurrences and signs of residual values have been considered, i.e. $\{k_0, k_1, k_2, k_3\}$, $\{-k_0, -k_1, -k_2, -k_3\}$ and $\{k_3, k_2, k_1, k_0\}$ are aggregated into one value.

Once constructed, the final feature vector $\boldsymbol{z}$, that characterize a given image is passed to the classification. The dimensionality of the feature vector for the Spatial

domain Rich Model (SRM) is 34671. Note that many features are useless and thus the feature vector can be improved to a reduced version of 32016 features [Boroumand and Fridrich, 2017]

### 5.4.2 maxSRM

To further improve the steganalysis performance against the ever more sophisticated steganography algorithms, more discriminating statistical features are in demand. One way to do that is to incorporate the extra information of *the clairvoyant scenario with side channel informed* (seen in Section. 5.3.2). One of the most well-known practical algorithms that go in this direction is the maxSRM [Denemark et al., 2014].

The maxSRM algorithm is built on the SRM algorithm and proceeds in the same way. The only exception comes from the computation of the co-occurrence features where the selection channel information is used Denemark et al. [2014]. For instance, taking the horizontal 4 neighbourhoods the co-occurrences given by:

$$C^{h}_{(k_0,k_1,k_2,k_3)} = \sum_{i,j} max(\beta_{i,j}, \beta_{i,j+1}, \beta_{i,j+2}, \beta_{i,j+3}) \times [(r_{i,j}, r_{i,j+1}, r_{i,j+2}, r_{i,j+3}) = (k_0, k_1, k_2, k_3)] \tag{5.9}$$

where $(\beta_{i,j}, \beta_{i,j+1}, \beta_{i,j+2}, \beta_{i,j+3})$ are the estimated probability of the embedding modification, which are computed using the cost map obtained from the image to be examined with the help of an optimal simulator (seen in Section. 3.3.4).

From Equation. 5.9, only the highest modification probability of the four consecutive pixels is used to contributes to the feature.

A more adapted version of maxSRM was later proposed in [Denemark et al., 2016b] which is called ad $\sigma$maxSRM. Compared to maxSRM, $\sigma$maxSRM replaces the maximum value of the estimated embedding modification probabilities with a

maximum of the expected difference in the filtered residual.

$$C^{h}_{(k_0,k_1,k_2,k_3)} = \sum_{i,j} max(\sigma_{i,j}, \sigma_{i,j+1}, \sigma_{i,j+2}, \sigma_{i,j+3}) \times [(r_{i,j}, r_{i,j+1}, r_{i,j+2}, r_{i,j+3}) = (k_0, k_1, k_2, k_3)] \tag{5.10}$$

where $\sigma_{i,j}$ is an estimated value of the expected difference in the filtered residual.

### 5.4.3 Ensemble classifier

The next step, after features extraction, is classification. For this purpose, a learnable classifier is used. The classifier relies on the relevant extracted features resulting from the first step to learn a separation function that can accurately discriminate the samples belonging to the cover or stego classes. Among the most famous classifiers in the steganalysis field, we have the Ensemble Classifier (EC) [Kodovský et al., 2012].

This classifier was proposed as an alternative to traditional classifier used for steganalysis purposes such as support vector machines (SVMs) [Hearst, 1998]. Indeed, modern steganalysis relies on increasingly more complex features space leading to a high training complexity, which makes SVM a non-adapted classification tool for modern steganalysis.

The Ensemble Classifier consists of many simple base learners (many Fisher Linear Discriminant), where each base learner represents a simple classifier which is independently trained. Thus, the training of the ensemble classifier consists of the training of its sub classifiers (FLDs). Each sub-classifiers is trained, like an ordinary classifier as seen in Section. 2.1, on randomly selected sub-spaces of the original feature space. The dimensionality of the random sub-spaces can be chosen to be much smaller than the full dimensionality $m$. This helps to significantly decrease the training complexity, thus allowing the use of more complex feature vectors.

Once all base learners are trained, we move to the next step, **prediction**. For a given example from the testing set, each base learners returns a score (prediction).

The final prediction is then obtained by aggregating these scores. Despite the fact that the individual performance of base learners may be poor, the final accuracy increases considerably once their results are merged and eventually stabilizes for a sufficiently large number of base learners. This strategy is known in the machine learning field as bootstrap aggregating or bagging [Breiman, 1996].

In the following, we provide a more formal description of the ensemble classifier implementation.

Let $D'_{train} = \{\boldsymbol{z}^{(i)}, y^{(i)}\}_{i\in[1,s_{train}]}$ be a training set, and $D'_{test} = \{\boldsymbol{z}^{(i)}, y^{(i)}\}_{i\in[1,s_{test}]}$ a testing set with $\boldsymbol{z}^{(i)} \in \mathbb{R}^m$ a feature vector that describes the $i$-th image, and $y_i \in \{0, 1\}$ the class associated with it (0 for a cover image, and 1 for a stego image). Let $B_l$, $l \in [1, L]$, with $L$ the number of base learners composing the EC.

The objective of the training phase is to train each base learner to correctly map the samples $\boldsymbol{z}^{(i)}$ from $D'_{train}$ to their correct class $\{0, 1\}$. To this end, and in order to reduce the computational complexity, the base learners' training is performed only on a $m_{sub}$-dimension feature subspace (with $m_{sub} < m$). In practice, before the training phase, each base learner $B_l$ pseudo-randomly chooses a subset of features ($m_{sub}$ features) from each feature vector $\boldsymbol{z}^{(i)} \in \mathbb{R}^m$.

For the $l$-th FLD classifier, the used training set is given as follow:

$$D'^{l}_{train} = \{\boldsymbol{z}^{(i)}, y^{(i)}\}_{i\in[1,s_{train}]} \tag{5.11}$$

with $\boldsymbol{z}^{(i)}$ is $m^l_{sub}$ feature vector.

The prediction given by this classifier for a given sample $\boldsymbol{z}^{(i)} \in D'^{l}_{train}$ is given as follow:

$$B_l\left(\boldsymbol{z}^{(i)}\right) = \begin{cases} 1 & \text{if } \hat{y}^{(i)} \geq T \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

with $T$ a learnable parameter represents the threshold, $\hat{y}^{(i)}$ the projection value returned by the base learner $B_l$.

During training, the value of the threshold of each FLD classifier is adjusted to minimize the total detection error ($P_E$) on the training data. In this spirit [Kodovský et al., 2012] have proposed to compute the probability of detection error $P_E$ as follows:

$$P_E = \min_{P_{FA}} \frac{P_{FA} + P_{MD}(P_{FA})}{2} \tag{5.13}$$

During the test phase, for a given sample $\boldsymbol{z}^{(i)} \in D'_{test}$, each FLD classifier returns a binary decision indicating the class assigned to the $i$-th image associated to $\boldsymbol{z}^{(i)}$ feature vector. The final decision is the obtained by merging the results of the different FLD classifiers (by majority voting) according to the following formula:

$$B_l\left(\boldsymbol{z}^{(i)}\right) = \begin{cases} 1 & \text{if } \sum_{l=1}^{L} B_l\left(\boldsymbol{z}^{(i)}\right) > L/2 \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

The use of ensemble classifiers was one of the main contributions in the steganalysis field pending the last ten years, as it permitted the use of more complex features that can capture many different dependencies without worrying about the dimensional restriction.

## 5.5 Steganalysis using one-step learning approaches

Steganalysis is not the only research field that depends significantly on the feature extraction step. In computer vision tasks, such as object detection, researchers consider that feature extraction is the keystone to the development of a reliable

object-detection framework. This implies that the used features should be robust to the objects different orientations, including scales, viewing angles and occlusions. However, it is almost impossible to design a handcrafted features vector capable of covering such complexity; therefore, the performance of this framework will remain far from optimal.

During the well-known image-net competition in 2012 [Russakovsky et al., 2015], a new method which can learn features by mathematical optimization rather than hand extraction was proposed by the winner of the competition. He proposed the use of a convolutional neural network (discussed previously in Section. 2.2.6), this network is known as ALexNet [Krizhevsky et al., 2012]. It has surprisingly outperformed by far all conventional feature-based approaches proposed by other teams. Since then, computer vision researches have shifted from the improvement of the feature extraction algorithm to the design of self-learning feature methods. In other words, the routine of making enormous efforts into manual feature extraction has been replaced by designing well-adapted neural network architectures that can automatically learn relevant features.

Deep learning, more specifically convolutional neural networks, is being embraced into the field of steganalysis in an attempt to potentially learn more relevant and effective features and thus improve performance. However, it is not surprising to find that a random designed CNNs usually can not converge when it is trained as a steganalyzer. Therefore, some customized designs specific to steganalysis are required in order to incorporate the domain knowledge into the learning of CNN based steganlyzer.

We illustrate in Figure. 5.6 the difference between the two-step learning approach and the one-step learning approach.

One can notice that the principle remains the same whether it is a one-step or a two-steps learning approach.

The first attempt to use a Deep Learning method for steganalysis dates back to 2014 [Tan and Li, 2014]. The authors used unsupervised learning with a stack of

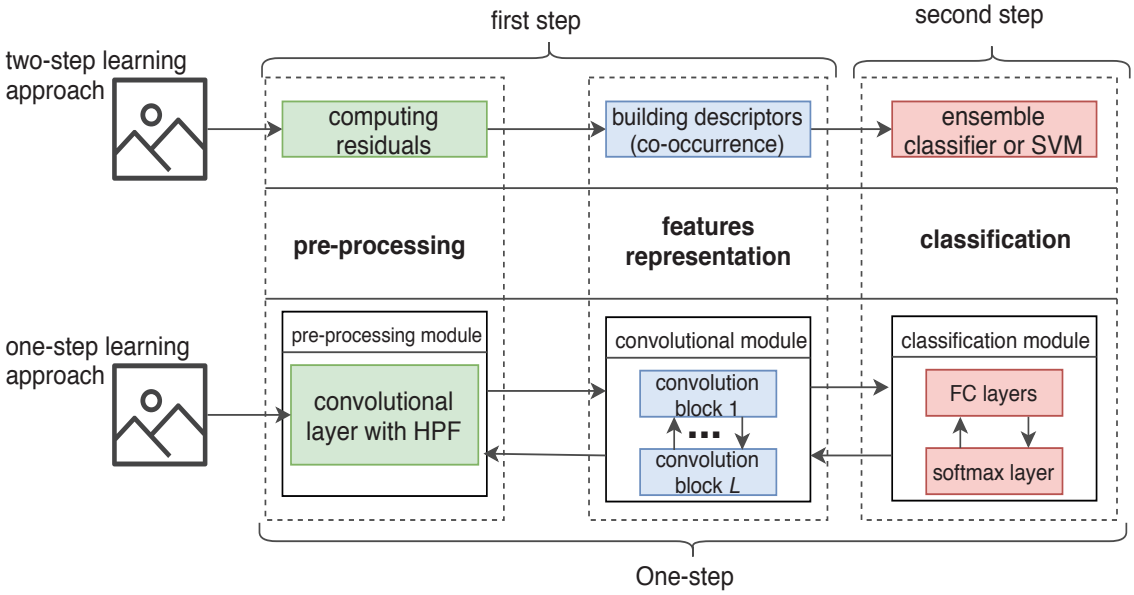



FIGURE 5.6: The two-step learning approach versus one step learning approach for steganalysis.

auto-encoders. One year later, Qian *et al.* [Qian et al., 2015] proposed to use, for the first time, supervised learning with Convolutional Neural Networks. In the pursuit of this research, Pibre has proposed another network architecture [Pibre et al., 2016]. In 2016, the first results, close to those of the state-of-the-art, were obtained with an Ensemble of CNNs [Xu et al., 2016b]. The Xu-Net [Xu et al., 2016a] CNN is used as *base learner* of the Ensemble of CNNs.

Other networks have been proposed in 2017, this time for JPEG steganalysis. In [Zeng et al., 2017a], the authors proposed a pre-processing inspired by the Rich Models, and the use of a big learning database. The results were close to those of the state-of-the-art in the frequency domain. In [Chen et al., 2017], the network is built with a *phase-split* inspired by the JPEG compression process. An Ensemble of CNNs was required to obtain results that were slightly better than those of the state-of-the-art. In [Xu, 2017], a CNN inspired by ResNet [He et al., 2016] with the "shortcut connection" trick and 20 layers also produced results that were slightly better than those of the state-of-the-art.

By the end of 2017 and until the present time, the studies are strongly concentrated on spatial steganalysis. Among the most important networks that have contributed

to the current steganalysis revolution, we find Xu-Net [Xu et al., 2016a], Ye-Net [Ye et al., 2017], Yedroudj-Net [Yedroudj et al., 2018b] (will be further reviewed in the contribution chapter), ReST-Net [Li et al., 2018], SRNet [Boroumand et al., 2019], and Zhu-Net [Zhang et al., 2019].

All Convolutional neural Networks designed for steganalysis purposes have more or less the same structure. They are mainly built in three parts, which we will call modules. Besides the two modules (convolution module, and the classification module), discussed previously in Section. 2.2.6, CNN-based steganalyzer includes another module called **pre-processing module**. This module is used as a preliminary filtering step that is intended to increase the signal-to-noise ratio (SNR) between the weak stego signal (if present) and the image signal, thereby allowing the network to converge more quickly while achieving good performance [Qian et al., 2015].

In the following, we will discuss the architecture of some of these networks, while presenting the components of their three modules.

### 5.5.1 Xu-Net

In early 2016, Xu [Xu et al., 2016a] proposed one of the most important networks in the field of steganalysis, which was then named Xu-Net. The importance of this network lies in the fact that it was the first network to achieve results comparable to those of the state-of-the-art based on the two-step learning approach (SRM+EC).

As an illustration, Figure. 5.7 schematizes the overall architecture of Xu-Net. The network accepts as input grey-scales images of size $512 \times 512$, and outputs a vector of two values called imprecisely "probabilities". Each value or probability represents how likely the inputted image belongs to a cover or stego class. The Xu-Net, like the most of other steganalysis CNN, is composed of a pre-processing module that is followed by a convolutional module and ends with a classification module.

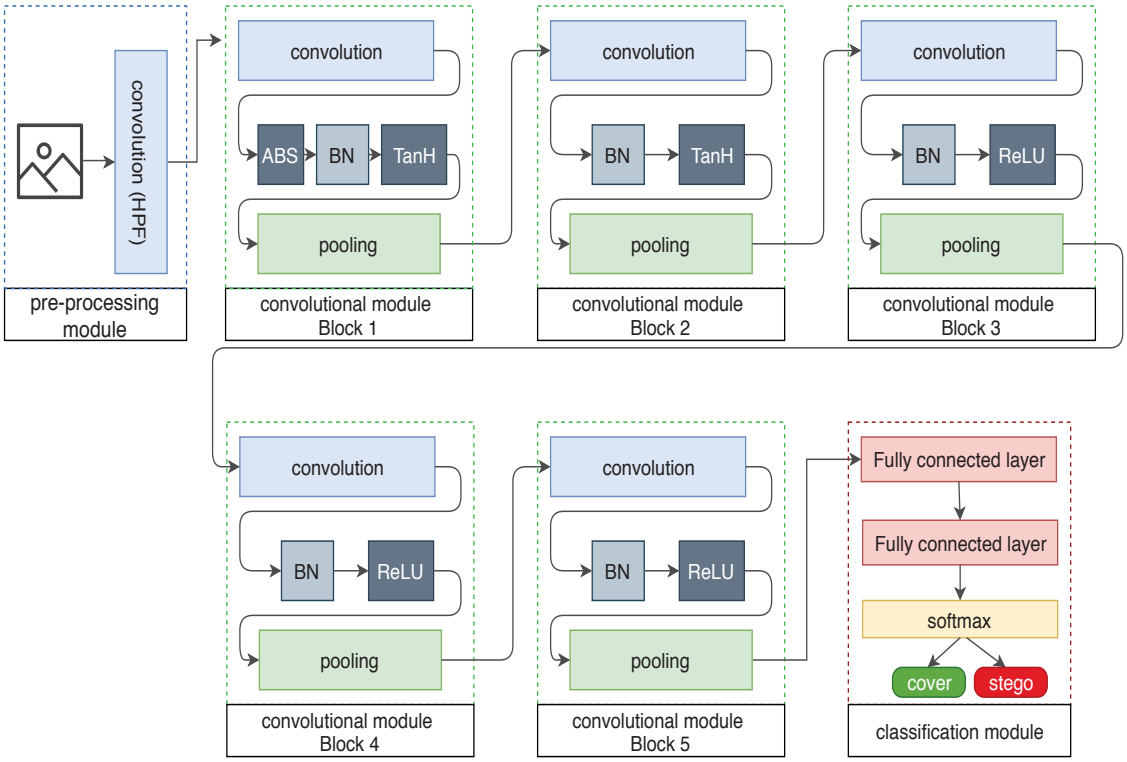



FIGURE 5.7: Xu-Net overall architecture.

#### 5.5.1.1 Xu-Net pre-processing module

The pre-processing module of Xu-Net, as illustrated in Figure. 5.7, consists of one convolutional layer (the first layer of the network $l = 1$). The kernel of this layer $k^{[1]}$ is initialized with a high pass filter kernel derived from SRM [Fridrich and Kodovský, 2012] and given below:

$$k^{[1]} = \frac{1}{12} \begin{bmatrix} -1 & +2 & -2 & +2 & -1 \\ +2 & -6 & +8 & -6 & +2 \\ -2 & +8 & -12 & +8 & -2 \\ +2 & -6 & +8 & -6 & +2 \\ -1 & +2 & -2 & +2 & -1 \end{bmatrix} \tag{5.15}$$

The parameters in this $5 \times 5$ kernel are fixed and not optimized during training.

With the help of Equation. 2.15, the input image $\boldsymbol{x}^{(i)} = (x_1, ..., x_n) \in \{0, ...255\}^n$, $n = [512 \times 512]$, is filtered using the $k^{[1]}$ filter, and transformed to noise residuals $F^{[1]}$ (in the form of a feature map) in order to boost the SNR.

The obtained feature map (filtered image) is then given to the convolution module in order to transform it into a set of features (feature vector).

#### 5.5.1.2 Xu-Net convolutional module

The convolutional module is composed of five blocks of layers; each block begins with a convolutional layer that produces feature maps and ends with an average pooling layer responsible for sub-sampling the processed feature maps. Between these two layers, we find a batch normalization layer and an activation layer. We provide, in Table. 5.1, the architecture, as well as the parameters utilized on each layer of this module.

All blocks include a convolution layer with a $stride = 1$; however, the size of the kernel changes from block to another. For example, the size of the 3D kernel used for the first layer is $8 \times 5 \times 5$, which corresponds to 8 kernels whose height and width are equal to 5. In the last three blocks, the spatial size of kernels is limited to $1 \times 1$ in order to limit statistical modelling.

As for the activation layer, first and second block are equipped with the hyperbolic tangent (TanH) seen previously in Section. 2. For the three other blocks, the ReLU is used instead. Exceptionally on block 1, an absolute activation layer (ABS) is used to force the network to take into account the symmetry (sign) existing in the noise residues. To reduce the risk of CNN training falling into poor local minimum, batch normalization (BN) is included in all blocks, and it is performed before each non-linear activation layer.

Average pooling layer, which is present in all four first blocks, reduces the size of the feature-map by a factor of two as the stride parameter is set to 2. In the final block, through global averaging, the pooling layer in Group 5 merges each spatial map to a single element (128 maps of size $32 \times 32$ to 128-D features). In this way, the whole CNN is prevented from grasping the location information of embedded pixels from the training data. The obtained 128-D feature vector is then used by the classification module to learn.

| Blocks | Layers | F-M input size | F-M output size |
|---|---|---|---|
| Block 1 | Convolutional layer<br>size: $8 \times 5 \times 5$ stride:1<br>Absolute value layer<br>Batch Normalization (BN)<br>Activation function : TanH<br>Average pooling<br>size: $5 \times 5$ stride:2 | $1 \times (512 \times 512)$ | $8 \times (256 \times 256)$ |
| Block 2 | Convolutional layer<br>size: $16 \times 3 \times 3$ stride:1<br>Batch Normalization (BN)<br>Activation function : TanH<br>Average pooling<br>size: $5 \times 5$ stride:2 | $8 \times (256 \times 256)$ | $16 \times (128 \times 128)$ |
| Block 3 | Convolutional layer<br>size: $32 \times 1 \times 1$ stride:1<br>Batch Normalization (BN)<br>Activation function : ReLU<br>Average pooling<br>size: $5 \times 5$ stride:2 | $16 \times (128 \times 128)$ | $32 \times (64 \times 64)$ |
| Block 4 | Convolutional layer<br>size: $64 \times 1 \times 1$ stride:1<br>Batch Normalization (BN)<br>Activation function : ReLU<br>Average pooling<br>size: $5 \times 5$ stride:2 | $32 \times (64 \times 64)$ | $64 \times (32 \times 32)$ |
| Block 5 | Convolutional layer<br>size: $128 \times 1 \times 1$ stride:1<br>Batch Normalization (BN)<br>Activation function : ReLU<br>Global Average pooling<br>size $32 \times 32$ stride:1 | $64 \times (32 \times 32)$ | $128 \times (1 \times 1)$ |

TABLE 5.1: Xu-Net convolutional module.

#### 5.5.1.3 Xu-Net classification module

The classification module is a shallow multi-layer Perceptrons for two-class classification problem (see Section. 2.2.3). This module is composed of a two fully-connected (FC) layer which the number of neurons is 256 and 2, respectively. A softmax activation function is used over the last fully connected layer. This is to normalize the scores delivered by the network between [0; 1]. These two scores

represent the probability of belonging to cover or stego classes. Final class label is determined by choosing the class corresponding to the larger score.

To resume, the key elements of this network are:

- The use of an absolute layer (ABS) after the first convolutional layer to facilitate and improve the statistical modelling taking into account the sign symmetry existing in the noise residuals,
- the use of BN layer which reduces the chances of falling to poor local minima while training the network,
- the use of the TanH activation function on the first two layers, thus limiting the range of data values and preventing the deeper layer from modelling larger values.

### 5.5.2 Ye-Net

By the end of 2017, the first network (as far as we know) that surpass the state-of-the-art methods on the tiny BOSS database [Bas et al., 2011], principally related to the old two-steps machine learning paradigm, was proposed by Ye under the name of Ye-Net [Ye et al., 2017]. This network came with two versions, the non-informed version and the side-channel-aware version. We give in Figure. 5.8 the overall architecture of this network.

Like the Xu-Net, the non-informed Ye-Net is composed of a pre-processing module, a convolutional module, and the classification module. However, while Xu-Net works on grey-scale images of size $512 \times 512$, Ye-Net is designed to steganalyze grey-scale images whose size is $256 \times 256$.

As illustrated in Figure. 5.8, the proposed CNN consists of 10 blocks and ends with a fully-connected layer with a 2- way softmax, which produces the distribution over 2 class labels. In the following, we give the overall architectures, and the used layers parameters of this network.

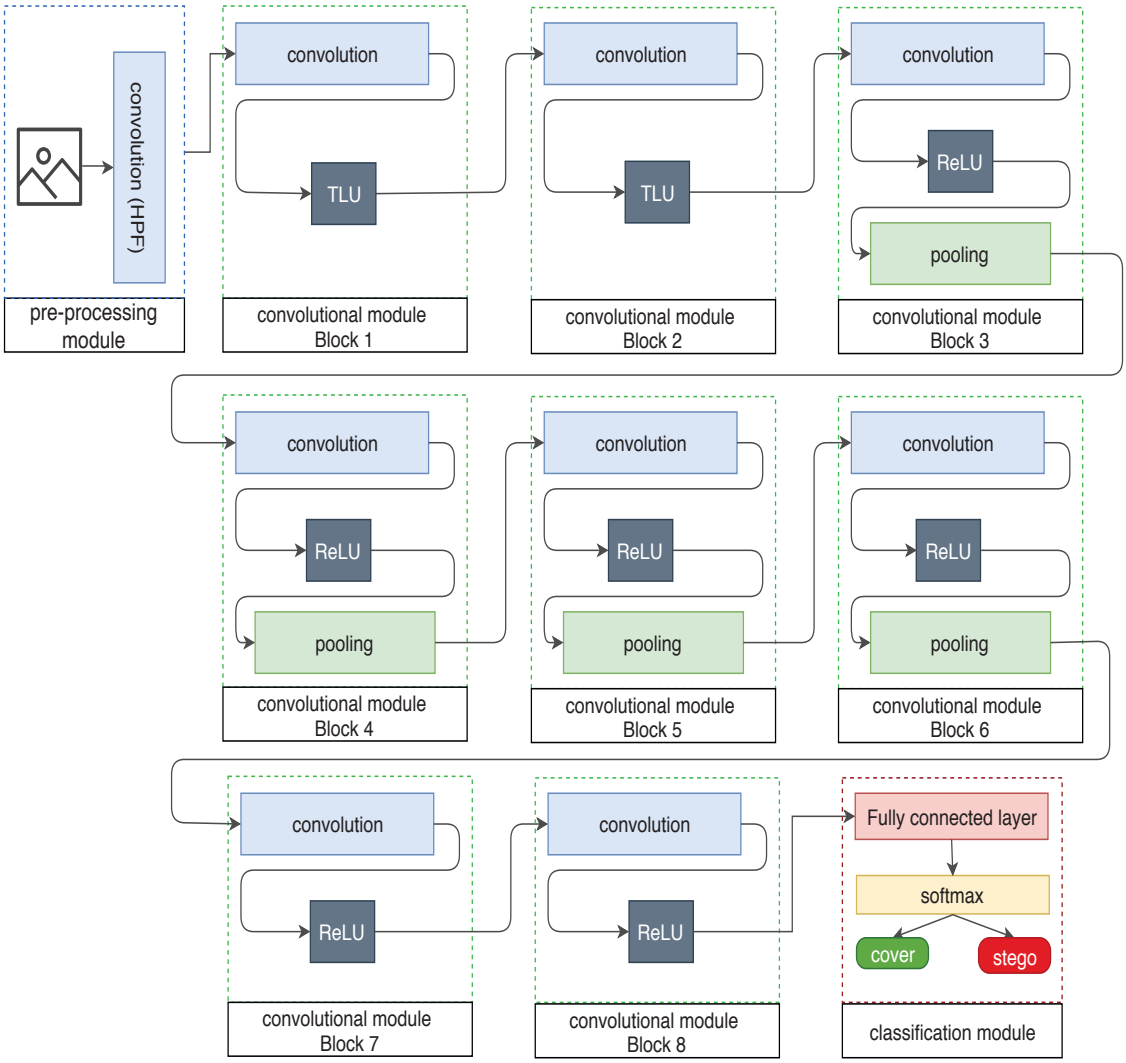



FIGURE 5.8: Ye-Net overall architecture.

#### 5.5.2.1 Ye-Net pre-processing module

The pre-processing module of Ye-Net is composed of one convolutional layer. Similarly to Xu-Net, Ye-Net authors have decided to initialize the weights of the pre-processing-convolutional layer (first layer) with high-pass filter kernels from the SRM instead of random values. However, rather than using one filter of the SRM, Ye proposed to incorporate all the 30 basic linear filters of the SRM, that is, the spam filters and their rotated versions.

This choice was justified experimentally by the fact that CNN is converging faster, and also by the network's better performance. As far as we know, SRM residuals help to improve the SNR (stego signal to the image content) and it is the diversity

provided by the combination of several high-pass filters (HPF) that makes the success of steganalysis-based rich models so successful. It was, therefore, logical to adopt the use of all SRM bank of filters within a CNN-based steganalyzer.

Unlike the Xu-Net, the kernels of the pre-processing module used by the Ye-Net are not fixed, but trainable. In other words, the kernels of the pre-processing layer are only initialized with the values of the SRM kernels. The final values of these kernels are to be learned during training. The filtered images outputted from this module are then transmitted to the convolution module.

#### 5.5.2.2 Ye-Net convolutional module

As illustrated in Tab. 5.8, the convolutional module CNN consists of 8 convolutional blocks. All 8 blocks include, at least, a convolutional layer and an activation layer. No Batch normalization is used. The pooling layer is suppressed from 1st to 3th, 7th and 8th block. For the remaining blocks, pooling layers with a stride equal to 2 are used.

Non-linear activation layer is included in each block and performed after the convolution layer. From the $3rd$ Block to the $8th$ Block the ReLU function is used for activation function. For blocks 1 and 2, a truncation function (given in Equation. 5.7) is adopted and referred to as "TLU".

At the end of the convolutional module, the outputted feature-map size is $16\times3\times3$. It is then flattened to 144-D feature vector, and transmitted to the classification module.

#### 5.5.2.3 Ye-Net classification module

Different from Xu-Net and other conventional CNN architectures that employ two (or more) fully-connected layers, the Ye-Net has only one fully connected layer composed of two neurons, and a softmax function, which produces the distribution

| Blocks | Layers | F-M input size | F-M output size |
|---|---|---|---|
| Block 1 | Convolutional layer size: $30 \times 3 \times 3$ stride:1<br>Activation Function : TLU | $30 \times (252 \times 252)$ | $30 \times (250 \times 250)$ |
| Block 2 | Convolutional layer size: $30 \times 3 \times 3$ stride:1<br>Activation Function : TLU | $30 \times (250 \times 250)$ | $30 \times (248 \times 248)$ |
| Block 3 | Convolutional layer size: $30 \times 3 \times 3$ stride:1<br>Activation Function : ReLU<br>Average Poling size: $2 \times 2$ stride:2 | $30 \times (248 \times 248)$ | $30 \times (123 \times 123)$ |
| Block 4 | Convolutional layer size: $32 \times 5 \times 5$ stride:1<br>Activation Function : ReLU<br>Average Poling size: $3 \times 3$ stride:2 | $30 \times (123 \times 123)$ | $32 \times (59 \times 59)$ |
| Block 5 | Convolutional layer size: $32 \times 5 \times 5$ stride:1<br>Activation Function : ReLU<br>Average Poling size $3 \times 3$ stride:2 | $32 \times (59 \times 59)$ | $32 \times (27 \times 27)$ |
| Block6 | Convolutional layer size: $32 \times 5 \times 5$ stride:1<br>Activation Function : ReLU<br>Average Poling size $3 \times 3$ stride:2 | $32 \times (27 \times 27)$ | $32 \times (11 \times 11)$ |
| Block 7 | Convolutional layer size: $16 \times 3 \times 3$ stride:1<br>Activation Function : ReLU | $16 \times (11 \times 11)$ | $16 \times (9 \times 9)$ |
| Block 8 | Convolutional layer size: $16 \times 3 \times 3$ stride:1<br>Activation Function : ReLU | $16 \times (9 \times 9)$ | $16 \times (3 \times 3)$ |

TABLE 5.2: Ye-Net convolutional module.

over 2 class labels. This is because the fully-connected layer involves too many parameters to be trained, hence take more time to learn.

#### 5.5.2.4 Ye-Net with side-channel aware

The performance of SCA-Ye-Net is increased compared to the non-informed Ye-Net, thanks to the incorporation of the channel selection knowledge (knowledge

of the probability of change of each pixel). The idea is to feed the non-informed Ye-Net, along with the image to be steganalyzed, the modification probability map.

The modification probability map is first filtered using another a convolutional layer equivalent to the one used on the pre-processing module of the non-informed version; however, its kernels values are replaced by their absolute values. The obtained filtered-modification probability map is then summed point-wise with the corresponding feature-maps generated by the pre-processing modules.

By incorporating the side-channel knowledge, it is possible to deliver more information about steganographic noise to the following convolutional layers and thus improve the performance of CNN.

The Ye-Net network can be summarized in the following point:

- The use of a set of trainable high-pass filters in the calculation of residual maps, whose values are initialized from the SRM filters,
- the use of a new activation function called TLU,
- the network is relatively deep compared to the already existing network dedicated to steganalysis,
- no Batch normalization is used which makes the network sensitive to parameters initialization,
- a second version which incorporates the side channel aware is suggested.

### 5.5.3 ReST-Net

In early 2018, a new CNN-based steganalysis named ReST-Net was proposed [Li et al., 2018]. Thanks to its special architectural design, this network has been able to surpass not only two-step learning approaches but also other CNN-based steganalysis methods (such as Ye-Net). As shown in the Figure. 5.9, the

particularity of this network relies on its structure. It consists of three parallel convolutional sub-networks and a fully connected classification module. The three sub-networks act as base learners in comparison to the ensemble classifier presented in Section. 5.4.3. Each sub-network accepts an input image of $512 \times 512$ and produces a 256-D feature vector. The three feature vectors are then concatenated and transmitted to the classification module to make the final decision. The classification module works by applying a weighted sum of the features provided by the different sub-networks.

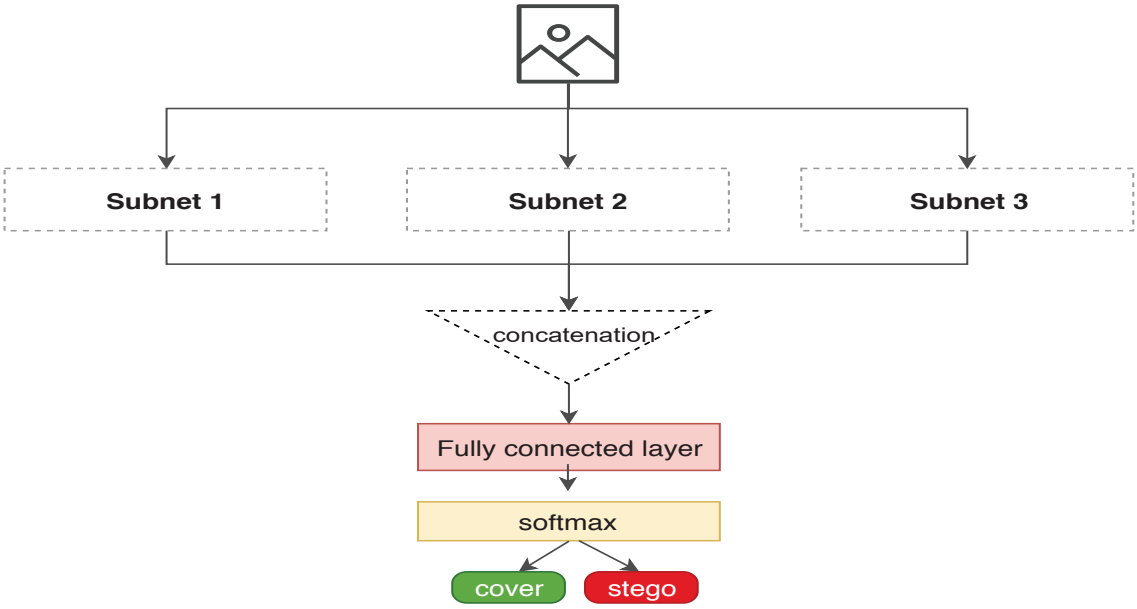



FIGURE 5.9: ReST-Net overall architecture.

The structure of the three sub-nets is identical, except for their pre-processing module. In the following, we give the overall architectures of the sub-network.

#### 5.5.3.1 ReST-Net pre-processing module

As pointed before, the three sub-networks have the exact same architecture except for the pre-processing module. This is because each sub-network is equipped with a different set of high-pass filters.

Instead of using only linear filters from SRM like Ye-Net, ReST-Net also employs the non-linear filters from SRM, together with Gabor filters [Song et al., 2015],

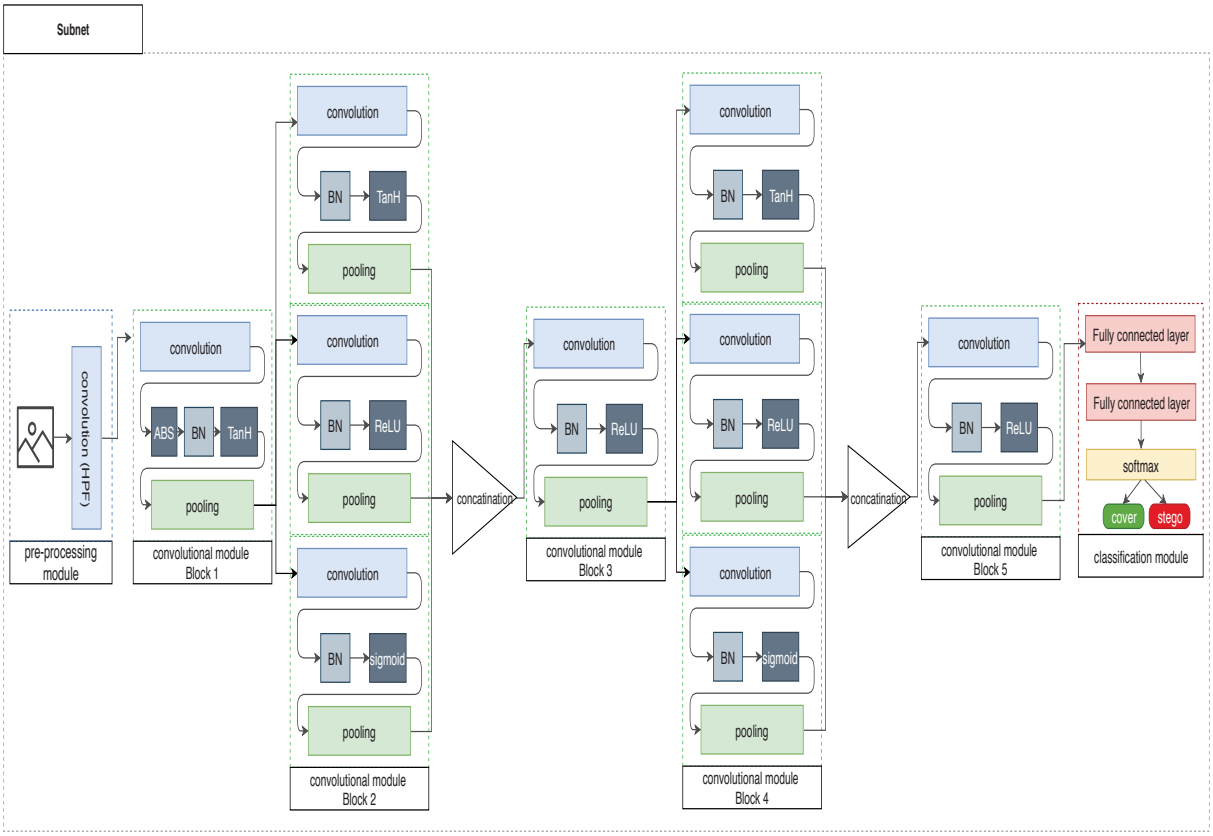



FIGURE 5.10: the overall architecture of a Sub-network of ReST-Net.

where each of these banks of filters is respectively employed in one of the three sub-networks as follow:

**Sub-net 1** The pre-processing module of the first sub-network is equipped with 16 Gabor filters of $6 \times 6$ to analyze the image with a specific frequency in a specific direction [Song et al., 2015].

**Sub-net 2** The second sub-network pre-processes the input image using a set of high-pass filters from the SRM [Fridrich and Kodovský, 2012]; the filters are padded with zeros to obtain a unified size of $5 \times 5$.

**Sub-net 3** The third sub-network also uses the SRM high-pass filters, however in order to introduce non-linearity, the resultant residual images are non linearly processed with "max" or "min" operation, as done in [Fridrich and Kodovský, 2012].

#### 5.5.3.2 ReST-Net convolutional module

The convolutional module of the ReST-Net sub-networks is composed of five convolutional blocks inspired from the Xu-Net described in Section. 5.5.1. The one major difference is the inclusion of what the authors call "Divers activation modules (DAMs) ". A DAM is formed by simultaneously using ReLU, Sigmoid and TanH activation functions on the same block. Then the resulting feature maps are concatenated and transmitted to the next convolutional block, as shown in Figure. 5.10. In Xu-CNN, TanH function is used over the first two convolutional blocks, while ReLU is used over the last three blocks. In ReST-Net sub-networks, the DAMS (displayed with coloured cells in Tab. ) replaces the TanH in the second block and the relu in the fourth block.

The authors justify the use of DAMs by the diversity that it offers. Indeed, the used activation functions respond differently to the traces of incorporation. Such diversity can, therefore, contribute to improving classification performance.

#### 5.5.3.3 ReST-Net classification module

Same as Ye-Net and unlike the Xu-Net, the ReST-Net employs only one fully connected layer over the classification module. This layer contains two neurons and followed by a softmax function. The output of this module is the prediction made by one sub-network (hence not the final decision). To this end, the ReST-Net is actually trained in two phases:

**first phase** each sub-network is separately pre-arranged to accurately classify a given image into its appropriate class (cover and stego). Once the pre-training is finished, the sub-network parameters are frozen, and the classification module is discarded,

**second phase** a new fully-connected layer involving 768 input neurons (256*3) is fed with the concatenated output feature vectors of the final convolutional blocks of the three sub-networks as shown in the Figure. 5.9. This

| Blocks | Layers | F-M input size | F-M output size |
|---|---|---|---|
| Block 1 | Convolutional layer<br>size: $24 \times 5 \times 5$ stride:1<br>Absolute value layer<br>Activation Function : TanH<br>Batch Normalization (BN)<br>Average Poling<br>size $5 \times 5$ stride:2 | $N \times (512 \times 512)$ | $24 \times (256 \times 256)$ |
| Block 2 | Convolutional layer<br>size: $32 \times 5 \times 5$ stride:1<br>Batch Normalization (BN)<br>Activation Function : TanH<br>Average Poling<br>size $5 \times 5$ stride:2 | $24 \times (256 \times 256)$ | $96 \times (128 \times 128)$ |
| | Convolutional layer<br>size: $32 \times 5 \times 5$ stride:1<br>Batch Normalization (BN)<br>Activation Function : ReLU<br>Average Poling<br>size: $5 \times 5$ stride:2 | $24 \times (256 \times 256)$ | |
| | Convolutional layer<br>size: $32 \times 5 \times 5$ stride:1<br>Batch Normalization (BN)<br>Activation Function : Sigmoid<br>Average Poling<br>size: $5 \times 5$ stride:2 | $24 \times (256 \times 256)$ | |
| Block 3 | Convolutional layer<br>size: $96 \times 1 \times 1$ stride:1<br>Batch Normalization (BN)<br>Activation Function : ReLU<br>Average Poling<br>size $3 \times 3$ stride:1 | $96 \times (128 \times 128)$ | $64 \times (64 \times 64)$ |
| Block4 | Convolutional layer<br>size: $64 \times 3 \times 3$ stride:1<br>Batch Normalization (BN)<br>Activation Function : TanH<br>Average Poling<br>size $5 \times 5$ stride:2 | $64 \times (64 \times 64)$ | $288 \times (32 \times 32)$ |
| | Convolutional layer<br>size: $64 \times 3 \times 3$ stride:1<br>Batch Normalization (BN)<br>Activation Function : ReLU<br>Average Poling<br>size $5 \times 5$ stride:2 | $64 \times (64 \times 64)$ | |
| | Convolutional layer<br>size: $64 \times 3 \times 3$ stride:1<br>Batch Normalization (BN)<br>Activation Function : Sigmoid<br>Average Poling<br>size $5 \times 5$ stride:2 | $64 \times (64 \times 64)$ | |
| Block 5 | Convolutional layer<br>size: $16 \times 1 \times 1$ stride:1<br>Batch Normalization (BN)<br>Activation Function : ReLU<br>Global Average Poling<br>size $32 \times 32$ stride:1 | $288 \times (32 \times 32)$ | $256 \times (1 \times 1)$ |

TABLE 5.3: ReST-Net convolutional module.

fully connected layer, together with a Softmax function serves as the final classification module.

The ReST-Net network can be summarized in the following point:

- a network composed of an ensemble of CNNs which that used as base learner,
- the use of two banks of non-trainable high-pass filters (Gabor, SRM)
- the use of a DAMs (different activation functions on the same block),
- a shallow network that is composed of three sub-networks (base-learners),
- the two-phase training which ensures stable and efficient convergence,
- the use of bootstrap aggregating or bagging but with an automatic and efficient way thanks to the fully connected layer which learns the best weights automatically.

### 5.5.4 SRNet

By the end of 2018, another network was proposed for steganalysis purposes. This network, thanks to its good performance, has became in a short period one of the most important CNN for steganalysis. It is named "Steganalysis Residual Network" and refereed as SRNet [Boroumand et al., 2019]. Among the interesting features of this network is the fact that it can be used for both spatial steganalysis and JPEG steganalysis. Like Ye-Net [Ye et al., 2017], SRNet can adopt the informed scenario (SCA) spatial steganalysis, however its philosophy is slightly different, as it is composed of only two parts: the convolution module and the classification module. Therefore, there is no pre-processing module. We give in Figure. 5.11 the overall architecture of this network.

As shown in the Figure. 5.11, SRNet consists of 12 convolutional blocks and ends with a classification module, which itself consists of one fully connected layer that is followed by a 2-way softmax, which produces the distribution on 2 class labels.

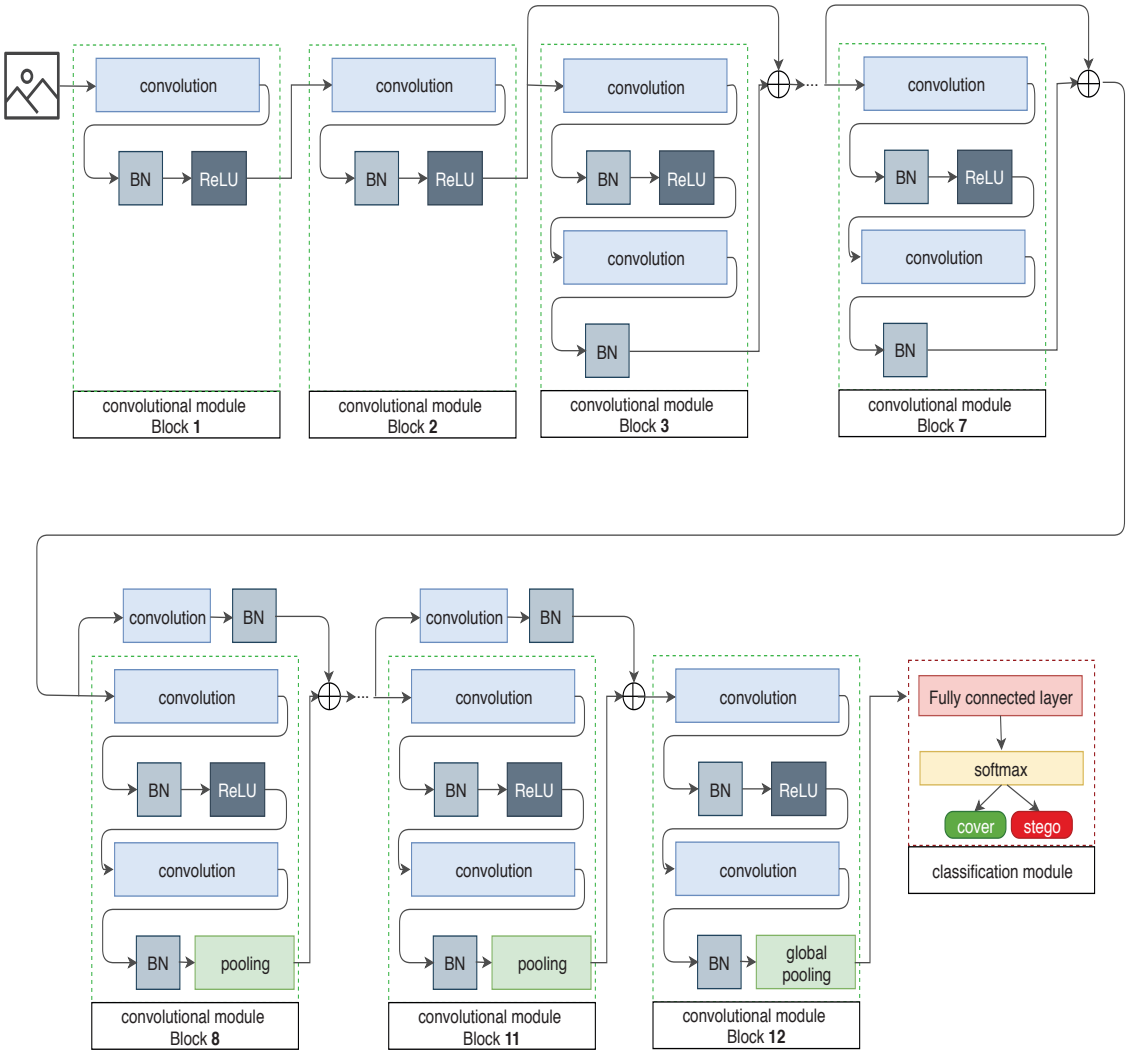



FIGURE 5.11: SRNet overall architecture.

In the following, we give the overall architectures, and the used layers parameters of this network.

### 5.5.4.1 SRNet pre-processing module

As already mentioned below, SRNet is not equipped with a pre-processing module. The authors claim that initializing first layer weights with high-pass filter kernels (preprocessing module) is not necessary for their network to converge, that is because SRNet can learn relevant filters used on the first layer. This may explain why this network can only run on large databases, as more examples are now needed to learn these filters, which are of great importance for network convergence.

#### 5.5.4.2 SRNet convolutional module

As shown in Figure. 5.11, SRNet convolutional module consists of 12 convolutional blocks. The two first blocks are composed of three layers. One convolutional layer followed by a batch normalization layer, and an activation layer (ReLU). Block 3 to Block 7 are slightly different, as they contain two convolutional layers and two batch normalization layer. In addition to this, there is the incorporation of the shortcut connections that heleps to propagate gradients to upper layers and avoid the problem of vanishing gradien that often negatively affects the convergence and performance of deep architectures.

As for Blocks 8 to 11, and on top of the 4 layers of Blocks 3 to 7, an average pooling layer is included. This layer is used to reduce the size of the feature-map. The shortcut connections of these blocks are also different, as they are provided by a convolution layer and a batch normalization layer (see Figure. 5.11). At the end of the $11-th$ convolutional block, the outputted feature-map size is $512 \times 16 \times 16$.

The last block (Block 12), is composed of 5 layers, two convolutional layers and two batch normalization layers and a global average pooling which reduces the 512 feature maps of dimension $16 \times 16$ to a 512-dimensional feature vector. The obtained vector is then fed to the classification module.

#### 5.5.4.3 SRNet classification module

Same as Ye-Net, the classification module of SRNet is composed of one fully connected layer of two neurons, and a softmax function, which produces the distribution over 2 class labels. This reduces drastically the number of the network parameters, hence accelerating the network's training.

#### 5.5.4.4 SRNet with side-channel aware (SCA-SRNet)

The selection channel has been incorporated in SRNet in the same fashion as in Ye-Net. They inject not only the image to be steganalyzed, but also the modification

probability map. The authors have declared that the performance of SCA-SRNet is increased compared to the non-informed SRNet, thanks to the incorporation of the channel selection knowledge (knowledge of the probability of change of each pixel).

The SRNet network can be summarized in the following point:

- A deep network that is composed of 12 convolutional blocks,
- no preprocessing module module, thus there is no initialization of the first layer weights,
- the use of a one activation function in all blocks which is ReLU,
- no pooling in the first four layers,
- the use of the residual shortcuts,
- a second version which incorporates the side channel aware is proposed,
- a network that works for both spatial and JPEG domains.

## 5.6 Conclusion

In this chapter, we have presented a state of the art of the main concepts of steganalysis. First, we defined the problem of detecting a hidden message, as well as the different categories of steganalysis attacks. Then, we reviewed the different class of steganalysis (Active, passive, malicious). We have then presented the targeted steganalysis methods and blind steganalysis methods. Next, we presented some possible steganalysis scenarios, which are defined according to the knowledge of the steganalysis. Then we have seen how algorithms belonging to the two-step-learning approach and one-step-learning approach operate. We ended this chapter with some practical algorithm of the two-step-learning approach and others of the one-step-learning approach.

# Part II

# Contributions

# Chapter 6

# YEDROUDJ-NET: An efficient CNN for spatial steganalysis

## Contents

## 6.1 Motivation

As mentioned in Section. 5.1, the objective of steganalysis algorithms is to minimize the probability of error for the classification of covers and stegos images. Until 2015, it was the era of two-step approaches. These approaches operate in two phases, first feature extraction and second, a classifier training. However, since the beginning of 2016, two-step approaches have been challenged by one-step approaches that rely on the use of deep learning, more specifically, convolutional neural networks.

These results were very encouraging, but when considering the gain obtained in other image processing tasks using Deep Learning [LeCun et al., 2015] methods, the steganalysis results, which are not even "10% better" than conventional approaches (Ensemble Classifier [Kodovský et al., 2012] with a Rich Model [Fridrich and Kodovský, 2012, Xia et al., 2017] or a Rich Model with a Selection-Channel Awareness [Denemark et al., 2014, 2016a]), are still very behind. In 2017, the main trends to improve CNN results were: using an ensemble of CNNs, modifying the topology by mimicking the Rich Models extraction process or using ResNet. In most of the cases, the design or the experimental effort is very high for a very small improvement of the performance.

By looking back to the *good practices* in deep learning as well as the recent studies in steganalysis (such as the use of a bank of filters, ABS layer, the batch normalization layer, truncation activation function ...), we experimentally designed a CNN for spatial steganalysis whose efficiency is naturally better than CNN appeared before mid- 2018 (when this work was published) [1]. This is performed without resort to either a design specific to the nature of images (spatial, jpeg, ...) or a CNN ensemble (which is known to improve the results). We focused on the design of the CNN, avoiding the use of tricks known to improve the performances such as transfer learning [Qian et al., 2016] or virtual augmentation of the database [Ye et al., 2017], etc. Additionally, the proposed network is not sensitive to the

[1] In 2019, SRNet [Boroumand et al., 2019] and Deng-Net [Deng et al., 2019] are probably the most efficient CNN for spatial steganalysis. Zhu-Net [Zhang et al., 2019], which is an update of Yedroudj-Net, will be published at the end of 2019 beginning of 2020

initialization of hyper-parameters and thus easily converges, which will be later discussed in the next Section. 6.2. We named this network the "Yedroudj-Net" CNN. It is compared with Xu-Net from Section. 5.5.1, and Ye-Net from Section. 5.5.2, and also with the Ensemble Classifier fed with the Spatial-Rich-Models for spatial steganalysis (previously presented in Section. 5.4.3 and Section. 5.4.1 respectively ).

## 6.2 Yedroudj-Net

Figure. 6.1 illustrates the overall architecture of our CNN. The proposed network follows the same stream of previous methods dedicated to steganalysis purposes. It is composed of a *pre-processing module*, five *convolutional blocks* forming the convolutional module, and a classification module made of three fully connected layers followed by a *softmax*. The network produces a probability distribution over the two class labels (cover, stego). The *pre-processing modules* filters the input

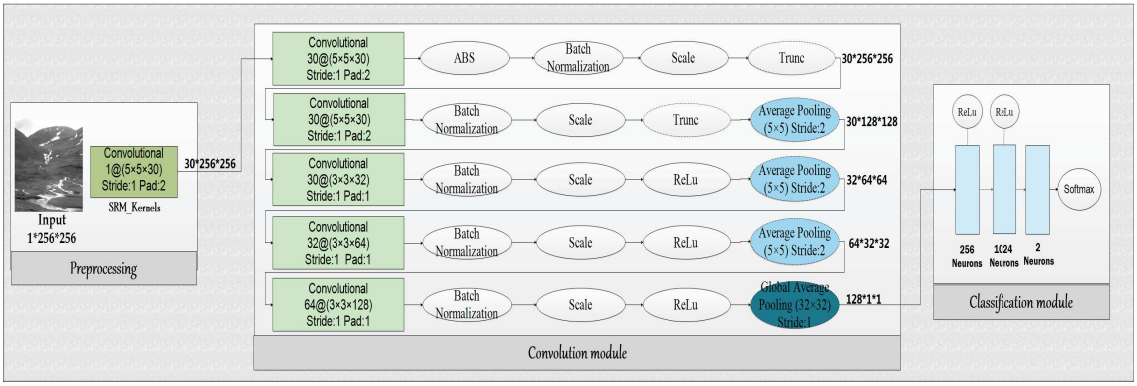



FIGURE 6.1: Yedroudj-Net CNN architecture.

cover/stego image with a high-pass filter in order to extract the noise component residuals. The pre-processed image then feeds the rest of the network. Previous studies [Qian et al., 2015, Pibre et al., 2016] observed that without this preliminary high-pass filter the CNN converges more slowly. This is because the pre-processing largely suppresses the image content, narrows the dynamic range, and thus increases the signal-to-noise ratio between the weak stego signal (if present)

and the image signal. As a result, the CNN can learn on a more compact and *robust* signal.

Inspired by the benefit of *diversity* [Fridrich and Kodovský, 2012], and similarly to Ye-Net (presented previously in Section. 5.5.2), we use the 30-basic high-pass filters from SRM [Fridrich and Kodovský, 2012], instead of using only one high pass filter such as in [Qian et al., 2015, Pibre et al., 2016, Xu et al., 2016a], in order to pre-process the input image. As a result, 30 outputs *feature maps* are produced. Note that the filters kernel values of the *preprocessing block*, i.e. the weights, are not optimized/learned during the training. This pre-processing has been integrated into a lazy fashion, directly into the CNN, such that the size of all kernels (weighting matrix) are set to 5×5. Their central part is initialized with the weights of the SRM kernels, and the remaining elements are padded to zero. No normalization of the kernels' values is performed.

The rest of our CNN consists in a convolutional module, dedicated to features representation, that transforms the input image into a *feature vector*, and a classification module, made of three fully-connected layers and a softmax layer, which produces the classification decision (cover or stego).

Similarly to Xu-Net, the convolutional module has five blocks marked as 'Block 1' through 'Block 5'. This convolutional module is intended to extract effective features for cover and stego images discrimination; see Figure. 6.1. Each block is made of the following succession:

1. a *Convolution Layer*. In the same spirit as Xu-Net, the size of the convolutional kernels is set to 5×5 for Blocks 1 and 2, and reduced to 3×3 for the Blocks 3 through 5. For all the convolution layers and similarly to Res-Net [He et al., 2016] and Xu-Net [Xu et al., 2016a], no biases are used. Biases terms are set to false on the convolution layer and moved to the Scale Layer.

2. an *Absolute Value* activation (ABS) layer. This ABS layer is only used in Block 1, similarly to Xu-Net, to force the statistical modelling to consider

the sign symmetry of the noise residuals. The relevance of this layer was observed in Xu-Net.

3. a *Batch Normalization (BN)*. The BN normalizes the distribution of each feature to a zero-mean and a unit-variance, and eventually, scales and translates the distribution. Practically, this normalization prevents small changes of the parameters from amplifying the gradient, as the data propagates through the network, and then spare the gradients from getting stuck in poor local minima. Moreover, the use of a BN layer desensitizes the training to the parameters initialization [Ioffe and Szegedy, 2015b], allows the use of a larger learning rate which speeds up the learning, and improves the detection accuracy [Chen et al., 2017]. From the above, the use of a BN layer is quite intuitive.

4- a *Scale layer*. Similarly to ResNet [He et al., 2016], and in contrast to Xu-Net [Xu et al., 2016a], who uses the BN layer to learn $\gamma$ and $\beta$ parameter in Equation. 2.22, we provide a BN layer with no ability of learning for these two parameters. Instead, they are learned by the independent *Scale Layer*. Our experiments, presented in the following Section. 6.3.4.2, showed that separating the zero-mean and unit-variance process (BN layer), from the shifting-mean and scaling-mean process (Scale layer) slightly increased the accuracy of the network.

4. a non-linear *Activation layer*. For the Blocks 1 and 2, the *Truncation function*, which formula is given in Equation. 5.7, is used to limit the range of data values and prevent the deeper layers from modelling large values. Indeed, these values are sparse and not statistically significant. This **outlier** suppression process can also be seen as the use of a *robustness function*. For the Blocks, 3 through 5, the classical *Rectified Linear Unit* (ReLU) is used because it yields good performances and its gradient computation is fast.

5. An *Average pooling* . The average pooling layer is exclusively used in Blocks 2 through 5. This layer allows to down-sample the feature maps, and thus reduces the dimensionality (as already explained in Section. 2.2.6.4). For

the last block, a *global* average pooling is performed to generate a one by one element for each corresponding feature map, thereby preventing the statistical modelling from grasping the location information of embedded pixels from the training data. There is no pooling in the first block to avoid information loss at the beginning of the network. Note that the suppression of the pooling layer (i.e. downsampling) from the firsts blocks allows preserving the stego signal [Boroumand et al., 2019].

The convolutional module ends with that global average pooling. The obtained (extracted) features are then fed to the classification module. The number of neurons in the first and second layers is set to 256 and 1024 respectively, while the last fully connected layer has only two neurons corresponding to the number of classes of the network's output. At the end of this module, a softmax activation function is used to produce a distribution over the two class labels.

### 6.2.1 Difference between the 3 CNNs

In this section we will briefly discuss the differences between our CNN *Yedroudj-Net* and the state-of-the-art CNNs for steganalysis in the spatial domain by the beginning of 2018 (*Xu-Net* CNN and *Ye-Net* CNN). In our comparisons, the *Xu-Net* has an architecture similar to the one of the original paper and demonstrated in Section. 5.5.1. Though, the size of its input images is set to 256×256 instead of 512×512. For that, we suppressed the *average pooling* from the first Block, which is a favourable measure since it avoids an early down-sampling. We also set a ReLU activation function among the Fully connected layers. As for the *Ye-Net*, we maintain the exact same architecture presented in Section. 5.5.2.

Figure. 6.3 shows the overall architectures of all CNNs. We summarize below the major similarities and differences between our proposed CNN and the two other CNNs:

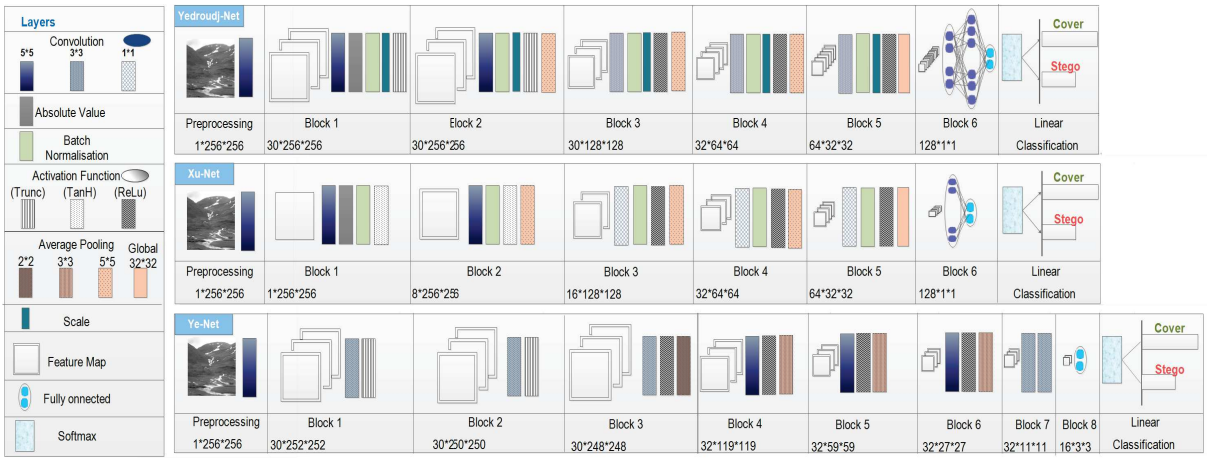



FIGURE 6.2: Comparison of Yedroudj-Net, Xu-Net, and Ye-Net architectures.

- Both Yedroudj-Net and Xu-Net use 5 convolution layers. Yedroudj-Net has nevertheless two times more features (256) at the input of the fully connected section. Ye-Net has more convolution layers.
- Both Yedroudj-Net and Xu-Net use a *Batch Normalization* layer (discussed in Section. 2.2.6.3); the Ye-Net does not.
- Both Yedroudj-Net and Xu-Net use the *Absolute Value* layer; the Ye-Net does not.
- Both Yedroudj-Net and Ye-Net use a 30 filter bank for pre-processing; the Xu-Net uses only one filter (the KV filter).
- Both Yedroudj-Net and Ye-Net Net use a Truncation activation function in Block 1 and 2 (We have found "Experimentally" that using Truncation activation function only in the Blocks 1 and 2 is the best choice in term of detection accuracy, those experiments are given in the next section); the Xu-Net does not.
- Yedroudj-Net has three (resp. Xu-Net two, and Ye-Net one) fully connected layer.

## 6.3 Experiments

### 6.3.1 Dataset and software platform

We use S-UNIWARD [Holub et al., 2014], and WOW [Holub and Fridrich, 2012], - two content-adaptive methods already presented in Section. 3.4- for the embedding in the spatial domain. We use the Matlab implementations with the simulator for the embedding and a random key for each embedding (online codes[2]). We thus avoid any wrong use of the C++ codes, i.e. a fixed and unique embedding key, as reported in [Pibre et al., 2016].

For comparative purposes, our steganalysis CNN, *Yedroudj-Net*, is put in competition with two of the state-of-the-art approaches in early 2018: *Xu-Net* CNN [Xu et al., 2016a], *Ye-Net* CNN [Ye et al., 2017], and with the hand-crafted feature set Spatial-Rich-Model [Fridrich and Kodovský, 2012] and the Ensemble Classifier [Kodovský et al., 2012] *SRM + EC*. For a fair comparison, all the involved steganalysis methods are tested on the same subsampled images from the BOSSBase database v.1.01 [Bas et al., 2011]. All CNNs experiments were performed with the publicly available *Caffe* toolbox [Jia et al., 2014] with necessary modifications, plus digits V5. All tests were run on an NVidia Titan X GPU card.

### 6.3.2 Training, Validation, Test

BOSSBase v1.01 is a widely known database that is used mainly in steganalysis. It consists of 10 000 grey-level images of size 512 × 512 coming from 7 different cameras.

Yet, due to our GPU computing platform and time limitation, we conduct all the experiments on images of 256×256 pixels. To this end, we re-sampled all the 512×512 images to 256×256 images, using the *imresize()* Matlab function with the default parameters. The 256×256 BOSSBase is then split into two sets, 50%

[2]http://dde.binghamton.edu/download/

(resp. the other 50%) of the cover/stego pairs are assigned to the testing (resp. training) set. 4000 out of the 5000 training set pairs are randomly selected for training; the remaining 1000 pairs are set aside for validation. The testing set is left untouched during the training stage.

During the CNNs training, we fixed a maximum of 900 epochs. Nevertheless, most of the time, we manually stopped the training when an over-fitting phenomenon appeared (usually before the epoch 200 for WOW and 300 for S-UNIWARD), i.e. when the *Loss* continues to decrease on the training set but starts to increase on the validation set. In practice, observing the *Loss* curve computed on the validation test leads us to keep two versions of the CNN: the CNN's models with minimum Loss (resp. maximum) on the validation set over the previous five epochs. Those two CNN's models are evaluated on the testing set, and we report the average error probability of detection for these two CNN's models.

For *SRM + EC* we use the SRM feature set of dimension=34 671 [Fridrich and Kodovský, 2012], and the Ensemble Classifier [Kodovský et al., 2012]. We report the minimum error probability under equals prior, averaged over 10 tests.

### 6.3.3 Hyper-parameters

We apply a mini-batch stochastic gradient descent (SGD) to train our CNN. The momentum is fixed to 0.95 and the weight decay to 0.0001. No dropout is used. The batch size in the training procedure is set to 16, due to GPU memory limitation (8 cover/stego pairs). All layers are initialized using **Xavier method**: the weights follow a Gaussian distribution and are chosen so that the variance for both input and output among each layer remains the same [Glorot and Bengio, 2010]. During the training, we use the *step policy* of Caffe to adjust the learning rate (initialized to 0.01). With this *policy*, each 10% of the total number of epochs, our learning rate is decreased by a factor gamma equal to 0.1. The threshold $T$, for the Truncations functions (see Equation. 5.7) is set to 3 for the first layer and

1 for the second layer, and the 30-basic high-pass SRM filters are not normalized. Note that the source codes and the materials files are available here [3].

### 6.3.4 Results without using any tricks

This section is divided into 2 subsections. In the first subsection we present a comparison w.r.t. The detection accuracy of our CNN, the two other CNN's (Xu-Net, Ye-Net) and the hand-crafted SRM+EC. These tests were carried out on BOSSbase v1.01, using two steganographic algorithms (WOW [Holub and Fridrich, 2012], S-UNIWARD [Holub et al., 2014]) with the embedding rates of 0.2 and 0.4 bpp.

In the second subsection, we present the experiment that is conducted to investigate the importance of each element of our CNN, such as the importance of using the 30 kernels of SRM, the scale layer effectiveness, the use of the Truncation activation function, and the fully connected module size.

#### 6.3.4.1 General performance comparisons

In Table. 6.1, we report the error probability obtained when steganalyzing WOW and S-UNIWARD embedding algorithms at 0.2 bpp and 0.4 bpp. The steganalysis methods are Yedroudj-Net, Xu-Net, Ye-Net, and SRM+EC [Kodovský et al., 2012, Fridrich and Kodovský, 2012].

For WOW algorithm, Yedroudj-Net has an error probability 8% lower (resp. 11%) at 0.2 bpp (resp. 0.4 bpp) compared to SRM+EC. The results are also favourable for S-UNIWARD steganalysis with an equal error probability at 0.2 bpp and 2% lower at 0.4 bpp.

Compared to the other CNN algorithms, our proposed CNN achieves far superior results. Yedroudj-Net is 2% to 6% better compared to Xu-Net for the two embedding algorithms and the two payloads. The results are even better when

[3] https://mehdi-yedroudj.wixsite.com/home/post/steganalys

| | ***BOSS 256×256*** | | | |
|---|---|---|---|---|
| | **WOW** | | **S-UNIWARD** | |
| **Steganalysis Payload** | 0.2 bpp | 0.4 bpp | 0.2 bpp | 0.4 bpp |
| SRM+EC | 36.5 % | 25.5 % | **36.6** % | 24.7 % |
| Yedroudj-Net | **27.8** % | **14.1** % | **36.7** % | **22.8** % |
| Xu-Net | 32.4 % | 20.7 % | 39.1 % | 27.2 % |
| Ye-Net | 33.1 % | 23.2 % | 40.0 % | 31.2 % |

TABLE 6.1: Steganalysis error probability comparison of Yedroudj-Net, Xu-Net, Ye-Net, and SRM+EC for two embedding algorithms WOW and S-UNIWARD at 0.2 bpp and 0.4 bpp.

compared to Ye-Net, where Yedroudj-Net is 3% to 9% better. Let us note that the two other CNNs are not always superior when compared to the SRM+EC. To beat SRM+EC, those approaches require using an ensemble of CNN, as proposed in [Xu et al., 2016b], or increasing the learning database, as proposed in [Zeng et al., 2017a, Yedroudj et al., 2018a], and showed in section below.

Note that extreme caution must be taken for the initialization of the learning rate of the Ye-Net and the management of its evolution through the epochs. Indeed, a bad initialization prevents the network from converging. In Yedroudj-Net and Xu-Net, the use of the Batch Normalization ensures less sensitivity to such a parameter setting.

To conclude on these general comparisons, in a classical clairvoyant scenario without any channel-awareness, and without using an ensemble, a larger database, a virtual augmentation of the database, or a transfer learning, Yedroudj-Net has a clear advantage over all the state-of-the-art methods (until 2018). For networks from 2018 or later, Yedroudj-Net remains a good net. Compared to ReST-Net, seen in 5.5.3, Yedroudj-Net is a smaller network that obtains close results (only 2-5% less efficient).

Compared to SRNet, Yedroudj-Net is only 1-7 % less accurate. However, it requires a much shorter learning time -This makes it the best choice when it comes to use in a GAN framework (see Section. 8.4.2)-. Moreover, Yedroudj-Net can converge on a very tiny database which is not the case for SRNet.

| Algorithms | Payload (bpp) | Ye-Net [13] | Yedroudj-Net [31] | SRNet [39] | Zhu-Net |
|---|---|---|---|---|---|
| WOW | 0.1 | 0.348 | 0.330 | 0.286 | **0.233** |
| | 0.2 | 0.230 | 0.206 | 0.190 | **0.131** |
| | 0.3 | 0.201 | 0.189 | 0.143 | **0.084** |
| | 0.4 | 0.177 | 0.158 | 0.099 | **0.065** |
| S-UNIWARD | 0.1 | 0.402 | 0.383 | 0.342 | **0.268** |
| | 0.2 | 0.318 | 0.305 | 0.228 | **0.171** |
| | 0.3 | 0.236 | 0.221 | 0.163 | **0.125** |
| | 0.4 | 0.157 | 0.171 | 0.134 | **0.081** |
| HILL | 0.1 | 0.396 | 0.386 | **0.353** | 0.357 |
| | 0.2 | 0.338 | 0.329 | 0.274 | **0.262** |
| | 0.3 | 0.263 | 0.247 | 0.225 | **0.204** |
| | 0.4 | 0.208 | 0.183 | 0.176 | **0.152** |

FIGURE 6.3: Steganalysis error probability comparison of Yedroudj-Net, Ye-Net, SRNet and Zhu-Net for different embedding algorithms. Extracted from [Zhang et al., 2019].

To our knowledge, best network in 2019 is Zh-Net. Note that Zhu-Net [Zhang et al., 2019] is an extension of Yedroudj-Net that beats SRNet and other networks from 2018 and later (including Yedroudj-Net).

#### 6.3.4.2 Additional experiments

In order to better understand why the Yedroudj-Net CNN is better than the other steganalysis methods, we conducted additional experiments to highlight the impact of its main components.

**A- SRM filter:** As already mentioned, Yedroudj-Net pre-process each input images with 30 high-pass filters extracted from SRM [Fridrich and Kodovský, 2012]. Note that analyzing an image by applying diverse filters is an idea that is known to improve steganalysis performances when using Rich Models [Fridrich and Kodovský, 2012].

Table. 6.2 confirms the benefit of using a set of high-pass kernels instead of using only one HPF. We can observe that equipping our CNN with all SRM kernels decreases the detection error by 3% on WOW at 0.2 bpp or 0.4 bpp.

| | WOW 0.2 bpp | WOW 0.4 bpp |
|---|---|---|
| Yedroudj-Net | 27.8 % | 14.1 % |
| Use only Kv filter | 30.9 % | 17.6 % |

TABLE 6.2: Evaluation of a filter-bank as a pre-processing. We report the error probability obtained when steganalyzing WOW [Holub and Fridrich, 2012] at 0.2 bpp and 0.4 bpp with the steganalysis method Yedroudj-Net CNN with or without the SRM filter-bank. When used without modification, the Yedroudj-Net CNN use the SRM filter-bank. Otherwise, we only use the Kv filter.

**B- Truncation function:**

| | WOW 0.2 bpp | WOW 0.4 bpp |
|---|---|---|
| Yedroudj-Net | 27.8 % | 14.1 % |
| Remove the second trunc (Tanh) | 29.5 % | 16.4 % |
| Remove both the trunc (Tanh) | 29.9 % | 16.1 % |
| Remove the second trunc (ReLU) | 30.1 % | 17.2 % |
| Remove both the trunc (ReLu) | 36.7 % | 21.4 % |

TABLE 6.3: Evaluation of different activation functions for the first and second blocks. We report the error probability obtained when steganalyzing WOW [Holub and Fridrich, 2012] at 0.2 bpp and 0.4 bpp with the steganalysis method Yedroudj-Net CNN with different activation functions. When used without modifications, the Yedroudj-Net CNN use the truncation activation function.

We use in our CNN the *Truncation* activation function proposed in [Ye et al., 2017]. To investigate its importance, we evaluate our CNN against WOW at 0.2 and 0.4 bpp with and without this activation function. The results are given in Table. 6.3 where the Truncation function of the second Block and both the first and second Blocks (see Figure. 6.1) is replaced by the ReLU or the Tanh activation function.

One can observe that when using the Truncation function instead of ReLu or Tanh, on both the first and second blocks, the decrease of the error probability, for ReLU, is 8% (resp. 7%) for WOW at 0.2 bpp (resp. 0.4 bpp), and 2% for Tanh whatever the payload. When using the Truncation function only on the first block and substituting it with Tanh or ReLu on the second Block, the decrease is around 2% to 3% whatever the substitution or the payload.

We guess that the Truncation function has the same role than as in Rich Models. It avoids treating features of rare occurrences, and it thus has a robustness effect. The ReLU is a drastic activation function because negative values are erased, but it is known to facilitate the CNN convergences. Strictly speaking, the Tanh do not "lose" any values, and at the same time introduce more robustness to outlier/rare features because it reduces their numerical influences. It seems that Truncation activation function is the most efficient trade-off between the ReLU and Tanh activation functions.

Note that authors in paper [Lu et al., 2019] have demonstrated the importance of the TRUNCATION activation function for both spatial and jpeg steganalysis. However, we believe that the gain of using the Truncation activation function depends on other factors (hyper-parameters) such us the gradient update or the network topology. For example, the SRNet only use ReLU but can obtain good results.

**Fully connected module**

| | WOW 0.2 bpp | WOW 0.4 bpp |
|---|---|---|
| Yedroudj-Net | 27.8 % | 14.1 % |
| Remove a layer from FC module | 29.9 % | 15.6 % |

TABLE 6.4: Evaluation of the Fully Connected Layers. We report the error probability obtained when steganalizing WOW [Holub and Fridrich, 2012] at 0.2 bpp and 0.4 bpp with the steganalysis method Yedroudj-Net CNN with or without a third layer. When used without modification, the Yedroudj-Net CNN has three fully connected layer. Otherwise, there is two layers.

For the Yedroudj-Net CNN, we use three Fully connected Layers in contrast to Xu-Net and Ye-Net (see Figure. 6.1). The results are reported in Table. 6.4 show how the probability of error is reduced (1%-2%) when augmenting the fully connected part from two to three layers. Remark that previous papers already have shown that this "classification" section of the CNNs is not the best, and for example, it can be improved if we cut the network, and replace the classification part with an SVM [Tang, 2013].

**Scale Layer (learn $\gamma$ and $\beta$)**

| | WOW 0.2 bpp | WOW 0.4 bpp |
|---|---|---|
| Yedroudj-Net | 27.8 % | 14.1 % |
| Remove the Scale Layer | 28.6 % | 15.0 % |

TABLE 6.5: Impact of using the Scale Layer. We report the error probability obtained when steganalyzing WOW [Holub and Fridrich, 2012] at 0.2 bpp and 0.4 bpp with the Yedroudj-Net CNN with or without the Scale Layer. Basically the Yedroudj-Net CNN uses the separate Scale Layer. When no Scale Layer is used, the shift and scale parameters are included in the Batch Normalization layer.

Table. 6.5 shows the performance of our CNN with and without the scale layers (see Figure. 6.1), applied on WOW at 0.2 and 0.4 bpp. Suppressing the scale layers means that the Batch Normalization assumes responsibility for learning both $\gamma$ and $\beta$. There is about 0.8% improvement in terms of error probability by using the Scale layer, which is a small improvement, but it is very easy to apply, and this justifies splitting the Batch Normalization into 2 layers.

| | WOW 0.2 bpp | WOW 0.4 bpp |
|---|---|---|
| 3×3 convolution | 27.8 % | 14.1 % |
| 1×1 Convolution | 27.8 % | 14.3 % |

TABLE 6.6: Evaluation of the Size of the kernels filters in the last convolutions (blocks 3 to 5). We report the error probability obtained when steganalizing WOW [Holub and Fridrich, 2012] at 0.2 bpp and 0.4 bpp with the Yedroudj-Net CNN steganalysis method with kernels whose size are 1×1 or 3×3 in the last convolutions. When used without modification, the Yedroudj-Net CNN has kernels of size 3×3 in blocks 3 to 5.

**Size of the kernels filters in the last convolutions** In our network, we use kernels of size 3×3 in the last blocks (Blocks 3 to 5; see Figure. 6.1). In Table. 6.6, we measure the impact of modifying the 1×1 kernels (same kernel size in Xu-Net) by 3×3 kernels for WOW at 0.2 and 0.4 bpp. The differences are not significant in term of error probability.

### 6.3.5 Results with a Base augmentation

Many tricks exist for improving the results of CNN (will be further discussed in next chapter). Among these tricks, there is the *base augmentation*. This option seems to be a very important measure to apply in order to better exploit the capacity of Deep Learning approaches.

| | BOSS | BOSS+BOWS2 | BOSS+BOWS2+VA |
|---|---|---|---|
| Yedroudj-Net | **27.8** % | **23.7** % | **20.8** % |
| Ye-Net | 33.1 % | 26.1 % | 22.2 % |
| Xu-Net | 32.4 % | 30.3 % | 30.5 % |

TABLE 6.7: Base Augmentation influence: error probability comparison of Yedroudj-Net, Xu-Net and Ye-Net on WOW at 0.2 bpp with a learning base augmented with BOWS2, and Virtually Augmented (VA).

In machine learning, and this is also true for CNNs, it is important to use a training base that is large enough to ensure a good generalization but also to avoid over-training. Some authors are prone to use big databases [Qian et al., 2015, Zeng et al., 2017a, Ye et al., 2017] in order to reach the state-of-the-art results. In the above experiment, we attempt to investigate the improvement brought by increasing the learning database size without modifying the testing set. It means that the learning set does not only contain images of the same kind as in the test set: e.g. the settings of cameras, the scenes of the learning set, can all be different from those of the testing set. We show the effects of increasing the image database on the error probability in Table. 6.7. To increase the size of our training set, BOWS2 database was employed [Bas and Furon, 2008]. This Database that was essentially created for a watermarking contest and consists of 10,000 8-bit grayscale 512×512-sized images. To this end, two scenarios have been tested.

In the first scenario, noted BOSS+BOWS2, we embedded the payload in the subsampled BOSSBase database v.1.01 [Bas et al., 2011]. We split this base into two sets: 50% of the cover/stego pairs to the training set, the rest to the testing set. Then, 10 000 additional pairs of cover/stego pair (obtained by subsampling BOWS2Base [Bas and Furon, 2008]) were added to the training set. The learning

database now contains 15 000 pairs of cover/stego images minus 1000 pairs from BOSS, set aside for validation.

In the second scenario, noted BOSS+BOWS2+VA, the database is virtually augmented by performing the label-preserving flips and rotations on the BOSS+BOWS2 training set. The size of the BOSS+BOWS2 training set is thus increased by a factor of 8, which virtually gives a final learning database made of 112 000 pairs of cover/stego images plus 1000 pairs from BOSS used for validation.

Table. 6.7 shows the performance comparisons in terms of detection error probability for Yedroudj-Net, Xu-Net [Xu et al., 2016a], Ye-Net [Ye et al., 2017], against the embedding algorithm WOW [Holub and Fridrich, 2012] at payload 0.2 bpp. For all algorithms, better performances are achieved using BOSS+BOWS2 compared to using only BOSSBase. The Yedroudj-Net obtains the best results and decreases its detection error probability by 4%. Ye-Net and Xu-Net respectively decrease their detection error probability by 7% and 2%.

When virtually augmenting the entire BOSS+BOWS2 learning set thanks to the 8 combinations of rotations and flips that do not introduce interpolation (i.e. BOSS+BOWS2+VA), the performances are again increased. The Yedroudj-Net keeps the best results and decreases its detection error probability by 7% (Ye-Net decreases it by 11%, and Xu-Net by 2%) compared to the case of only using BOSSBase for the training. Comparing to SRM+EC [Kodovský et al., 2012, Fridrich and Kodovský, 2012], whose error probability is 36.5% with a learning on the BOSSBase, the Yedroudj-Net obtain an error probability of 20.8% which give an improvement of 16%. The Ye-Net obtains an improvement of 14% and the Xu-Net an improvement of 6%.

These tests reveal how important it is to have a large database when using CNN of 5-7 blocks. The number of parameters (without taking into account the BN and/or scale) goes approximately from 50 thousand (Xu-Net) to 500 thousand (Yedroudj-Net). Such a huge number of unknown requires bearing enough samples. The experiments show that the CNNs still do not have enough learning samples. For a steganalysis of BOSSBase with CNNs of 5-7 blocks, even 112 000 pairs of images

(BOSS+BOWS2 virtually augmented) is not enough. Consequently, using a bigger base allows our CNN to achieve better performances even if the convergence time increases.

For instance, with a GPU card of the previous generation (Nvidia TitanX) on an Intel Core i7-5930K CPU 3.50GHz×12 with 32G of RAM, the convergence takes less than one day for learning Yedroudj-Net CNN on BOSSBase, three days on BOSS+BOWS2, and more than seven days on BOSS+BOWS2+VA.

At this point, it is clear that augmenting the training set improves the obtained results; however, it is not clear if the improvement is only due to a lack of data or also because the additional images came from the same cameras. We have nevertheless conducted additional experiments, reported in the next chapter, and it seems that in order to improve the performance, one must increase the database with images coming from the same sources and with a development process respecting the pixels resolutions and ratios.

### 6.3.6 Results with an ensemble of CNN

The other trick, which is widely used in deep learning to improve a neural network model's performance, is the usage of an ensemble of CNNs. Same as on ReST-Net (seen in Section. 5.5.3, we use three parallel sub-networks. All sub-networks adopt Yedroudj-Net's architecture; thus, we named it Yedroudj-Net$_{ensemble}$.

To train Yedroudj-Net$_{ensemble}$, we first start by training, separately, each of the three sub-networks on the exact same leaning set. The parameters of each network are initialized differently (convolution weights). Once all sub-networks are trained, we dispose of their fully connected parts and concatenate the output features of their convolutional modules into one feature vector.

Next, on top of these three subnets, we add another fully connected module (as shown in Figure. 5.9). It takes as input the feature vector and trained to minimize classification error. We use the same learning set as the one used to train the three subnets.

Once Yedroudj-Net$_{ensemble}$ converges, we test it on the remaining part of the BOSS base (the test set). The results are presented in Table. 6.8.

| | ***BOSS 256×256*** | | | |
|---|---|---|---|---|
| | **WOW** | | **S-UNIWARD** | |
| **Steganalysis Payload** | 0.2 bpp | 0.4 bpp | 0.2 bpp | 0.4 bpp |
| Yedroudj-Net | **27.8** % | **14.1** % | **36.7** % | **22.8** % |
| Yedroudj-Net$_{ensemble}$ | 24,3 % | 10.8 % | 33.1 % | 17.2 % |

TABLE 6.8: Evaluation of the efficiency of the Yedroudj-Net ensemble version. We report the error probability obtained when steganalizing WOW [Holub and Fridrich, 2012], S-UNIWARD [Holub et al., 2014], at 0.2 bpp and 0.4 bpp.

Compared to the scenario where only one network is used (noted Yedroudj-Net), the use of three networks gives better results. For WOW, we can notice that an improvement of 3-4 % is obtained when using Yedroudj-Net$_{ensemble}$. As for S-UNIWARD, Yedroudj-Net$_{ensemble}$ is better 3-5% compared to Yedroudj-Net.

## 6.4 Conclusion

In this chapter, we have presented Yedroudj-Net CNN, which is essentially designed for spatial steganalysis. This CNN gathers some recent design propositions in order to build a simple but efficient approach that beats the state-of-the-art approaches (until mid 2018) in a classical clairvoyant scenario without knowledge of the selection channel.

The key to the steganalysis performance improvement is the combination of the following elements: a bank of filters for the pre-processing step, a robust activation function, and a normalization associated with a scale Layer.

Additional experiments were carried out to test whether Yedroudj-Net's performance can be further improved when using some tricks. First, we evaluated the data augmentation and its impact on the Yedroudj-Net performance. Results have shown that by adding BOWS2 and virtually augmenting the learning database, the results become extremely satisfactory. An experiment on WOW at 0.2 bpp led to an error probability decrease of 16% compared to the RM+EC.

The other trick we tested was the use of an ensemble of CNNs. For this purpose, we have designed Yedroudj-Net$_{ensemble}$, the ensemble version of Yedroudj-Net, that is composed of 3 CNNs. The results show a decrease in error probability detection of nearly 4% for WOW (resp. 5% for S-UNIWARD) compared to the network Yedroudj-Net.

# Chapter 7

# How to augment a small learning set for improving the performances of a CNN-based steganalyze?

## Contents

## 7.1 Motivation

Convolutional neural networks (CNN) became very popular to solve classification problems in the last five years. Several authors have proposed to use CNNs to solve steganalysis problems [Qian et al., 2015], [Pibre et al., 2016], [Xu et al., 2016a], [Ye et al., 2017]. These methods yield encouraging results but remained comparable to the state-of-the-art algorithms performances. Authors have explored many approaches to obtain the best CNN-based steganalysis method, such as using a phase split [Chen et al., 2017], an ensemble of CNN [Xu et al., 2016b], the transfer learning [Qian et al., 2016] or the augmentation of the database [Ye et al., 2017], [Zeng et al., 2017b].

Let us now put aside the quest of the best deep learning network architecture for the steganalysis task. In this chapter, our objective is to look at a "real-world" problem [Ker et al., 2013], which is to learn with a small size database a good CNN model for steganalysis. This problem is also known as low regime learning. It is well-known that supervised approaches based on the use of CNNs need a lot of samples when used for steganalysis purposes. The number of images for the learning has even reached five millions of samples in [Zeng et al., 2017b]. The seminal propositions of Qian [Qian et al., 2015] and Pibre [Pibre et al., 2016] used from 8 000 to 80 000 spatial images resized to 256×256 (BOSSBase [Bas et al., 2011] or ImageNet [Krizhevsky et al., 2012]). In 2017 the authors mainly use around 5 000 pairs of images [Xu et al., 2016a], [Ye et al., 2017], [Chen et al., 2017], [Xu, 2017], which is probably insufficient. Moreover, in an operational and realistic protocol, the number of available images for the learning task could be much smaller than what is used in "laboratory".

Because all the CNN-based steganalysis algorithms are sensitive to the cover-source mismatch phenomenon [Cancelli et al., 2008, Ker and Pevny, 2014, Kodovskỳ et al., 2014], each time the source distribution is modified, the learning process has to be restarted. The aim of the study, presented in this chapter, is thus to look at the impact of *artificial data-augmentation*, which is probably more realist than having access to a huge database of a given source distribution. In all cases,

using data-augmentation is an automatic process which requires less human time consumption than searching for images of similar distributions.

Today, the classical scenario used to test an embedding algorithm efficiency is to use the BOSSBase [Bas et al., 2011] for training and testing, assigning 5000 of the 10000 images to the learning database, while the rest used as testing database. A classical way to artificially increase the learning database without changing the labels is to flip and rotate the learning database without interpolation [Krizhevsky et al., 2012].

Recently, Ye *et al.* [Ye et al., 2017] proposed to increase the size of the training database, by adding to the initial 50% of BOSSBase, the whole BOWS2 [Bas and Furon, 2008] database (this gives a total of 15000 pairs of images for the training set), while the test set is unchanged and is made of the remaining 50% of BOSS-Base. This process effectively improves the results in terms of error probability of detection. However, it could be considered as *a very lucky measure* because the improvement is essentially due to the fact that BOSSBase and BOWS2 share some identical camera models, and a similar "development" process[1].

The question is thus still open: how should we process in order to enrich a learning database? Can we enrich even more the BOSS learning base in order to obtain a huge learning base, and thus improve the steganalysis results? In this chapter, we intend to experimentally explore efficient ways to **increase** the learning database of a CNN based steganalyzer. In Section 7.2, we describe the experimental protocol and briefly present all the setups. In Section 7.3, we experimentally explore the different augmentation methods, and we draw conclusions on the practical question of the learning database augmentation.

[1]The "development" stands for the numerical processes transforming a colour RAW image to a 256×256 8-bit grey-levels image [Borghys et al., 2018]

## 7.2 Experimental methodology

In this chapter, the study on the data augmentation for spatial steganalysis is conducted only on the Yedroudj-Net presented in Chapter 6.

### 7.2.1 Objectives and Dataset baseline

Our final objective is to **increase the size of the learning database** of a CNN based steganalysis through data-augmentation in order to improve its performances. Indeed, increasing the number of learning samples is often beneficial for learning efficient features dedicated to a specific task. But, for steganalysis, the samples have to be selected carefully. The "new" samples have to share a "similar distribution" compared to the "original" samples. One thus tries to find **distribution-preserving transformations** which, when applied on an input cover or precover image, generate synthesized images that follow the same distribution. Those synthesized images could then be integrated into the learning database as additional images in order to increase the CNN classifier efficiency.

In this chapter, first, we explore the factors that are influencing a cover distribution such as the camera model, or the development, and second, we propose *distribution-preserving* transformations that allow to enrich an initial database and to improve the CNN efficiency.

Our baseline setup will thus be the same as the one presented in Section. 8.5.2, where the BOSSBase split into two sets. We assign 50% of the cover/stego pairs to the "original" training set, and the rest, to the testing set. **Regardless of the learning database enrichment, the test database will always contain images from and only from BOSSBase**. For a fair comparison, we will use the same test base for all the experiments. To summarize, the learning set will always contain at least 4000 pairs of BOSSBase images, and the **validation set will always contain 1000 pairs of BOSSBase images**.

### 7.2.2 Software platform

Same as in chapter 6, we used S-UNIWARD [Holub et al., 2014], and WOW [Holub and Fridrich, 2012] for the embedding in the spatial domain. All experiments were performed with the publicly available *Caffe* toolbox [Jia et al., 2014] with necessary modifications, plus digits V5. All tests were run on an NVidia Titan X GPU card.

### 7.2.3 Datasets

All the experiments are conducted on images of 256×256 pixels. We resampled all the 512×512 images to 256×256 images, using the *imresize()* Matlab function with the default parameters (bicubic interpolation with anti-aliasing).

For the various experimental setup, we are using the different databases listed below, and convert them to 256 × 256 images:

- the BOSSBase v1.01 [Bas et al., 2011] consisting of 10 000 grey-level images of size 512 × 512, never compressed, and coming from 7 different cameras,
- the BOWS2 [Bas and Furon, 2008] consisting of 10 000 grey-level images of size 512 × 512, never compressed, and whose distribution is close to BOSS-Base,
- the LIRMMBase [Pibre et al., 2016] consisting of 9 388 grey-level images of size 512 × 512, never compressed, and coming from 7 different cameras. All the used cameras are different from those used in BOSSBase.
- the PLACES2 [Zhou et al., 2017] containing more than one million of JPEG images coming from unknown cameras. For the experiments, those images are decompressed and then converted in grey-level and then resized.

For some experiments, we re-run a *development* process. For that, we use the *ImageMagick* a free, open-source software.

During the CNNs training, we regularly observe the *Loss* and *Accuracy* curves, computed on the validation test, to stop the training when an over-fitting phenomenon appears manually. This over-fitting occurs when the *Loss* curve continues to decrease in the training set but starts to increase on the validation set. For all the experiments, we report the error probability evaluated on the testing set.

### 7.2.4 Description of the different experimental setups

Below, we briefly listed all the experimental setups with a small description explaining each choice:

- **Setup 1: Classical enrichment**. In this setup, the goal is to obtain the performance baseline. The enrichment of the *original* learning database (made of 4000 pairs) is obtained thanks to the virtual augmentation using the label-preserving flipping and rotations [Ye et al., 2017], and the enrichment with BOWS2 images. This experiment is presented in Section 7.3.1,

- **Setup 2: Enrichment with other cameras**. In this setup, the goal is to evaluate the gain/loss of adding images from different cameras from the ones used in the *original* learning set. This experiment is presented in Section 7.3.2,

- **Setup 3: Enrichment with strongly dissimilar sources and unbalance proportions**. In this setup, the goal is to evaluate the gain/loss of adding a huge number of images generated using cameras and a development, totally different from those used in the *original* learning set. This experiment is presented in Section 7.3.3,

- **Setup 4: Enrichment with the same RAW images but with a different development**. In this setup, the idea is to evaluate the gain/loss of adding the same original RAW images whose development is different from the one used for the *original* learning set. This experiment is presented in Section 7.3.4,

- **Setup 5: Enrichment with a re-processing of the learning set**. In this setup, we assume that we don't have access to the RAW image. Thus, the objective is to evaluate the gain/loss of adding the same original images which are *re-processed*. This experiment is presented in Section 7.3.5,

## 7.3 Results and discussions

### 7.3.1 Setup 1: Classical enrichment

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.6** % |
| BOSS+VA | 24.2 % | 34.8 % |
| BOSS+BOWS2 | 23.7 % | 34.4% |
| BOSS+BOWS2+VA | 20.8 % | 31.1 % |

TABLE 7.1: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with and without Data Augmentation..

In Table. 7.1, we report the results with no enrichment (noted **BOSS**), the results with the Virtual Augmentation (VA) of the BOSS's training set (noted **BOSS + VA**; Virtual Augmentation consists in label-preserving flipping and rotations), the results with BOWS2 enrichment (noted **BOSS + BOWS2**), and the results with BOWS2 enrichment + the Virtual Augmentation (noted **BOSS+BOWS2+VA**). Some of these results have already been given in chapter 6. Note that for BOSS+BOWS2, the training set is made of 14 000 pairs (without counting the validation), 32 000 pairs for BOSS+VA (without counting the validation), and for BOSS+BOWS2+VA, the training set is made of 112 000 pairs (without counting the validation).

When the enrichment is obtained by only applying a virtual augmentation (BOSS+VA), a significant improvement is observed. The decrease of the error probability detection is 3% for WOW (resp. 2% for S-UNIWARD). This enrichment measure was initially proposed in [Krizhevsky et al., 2012], and it is indeed very efficient.

The reader should understand that the VA is an easy and low-cost operation in order to significantly improve the CNN performances.

One can also observe better performance when using BOSS+BOWS2 compared to only using BOSSBase. The CNN decreases its detection error probability by 4% for WOW (resp. 2% for S-UNIWARD). As stated in the introduction, BOSSBase and BOWS2 share some identical camera models and a similar "development" process. As also observed in Section 7.3.4, in a close setup, this enrichment setup ("similar cameras" + "similar development") allows to increase the performances. We guess that in that case, the added images increase the generalization capability of the network.

When the enrichment is obtained with BOSS+BOWS2+VA, again a significant improvement is observed. The decrease of the error probability detection is 7% for WOW (resp. 5% for S-UNIWARD) compared to the no-enrichment setup. Note that the results given in the current Section will be the reference performances for the comparisons given in the next sections.

The observations given in this Section are confirming that if the database augmentation ensures a good diversity of the database, the CNN can improve its detection accuracy. The experiments described in the next sections are thus done in order to understand better the properties that have to be kept when adding images to the *original* database.

### 7.3.2 Setup 2: Enrichment with other cameras

In Table. 7.2, we report the results with *no enrichment* (noted **BOSS**), the results with LIRMM enrichment (noted **BOSS + LIRMM**), the results with LIRMM and BOWS2 enrichment (noted **BOSS + LIRMM + BOWS2**), and the results with LIRMM and BOWS2 enrichment + the Virtual Augmentation (noted **BOSS + LIRMM + BOWS2 + VA**). Note that for *BOSS+LIRMM*, the training set is made of 14 000 pairs, for *BOSS+LIRMM+BOWS2*, the training set is made of 23

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.7** % |
| BOSS+LIRMM | 29.9 % | 38.6 % |
| BOSS+LIRMM+BOWS2 | 26.8 % | 36.9 % |
| BOSS+LIRMM+BOWS2+VA | 25.7 % | 36.1 % |

TABLE 7.2: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with a learning base augmented with either LIRMM, LIRMM+BOWS2, or LIRMM+BOWS2+VA. .

388 pairs (without counting the validation), and for the *BOSS+LIRMM+BOWS2*, the training set is made of 187 104 pairs (without counting the validation).

One can observe that results are worst when using *BOSS+LIRMM*, compared to only using *BOSSBase*. There is 2% increase of the detection error probabilities for both WOW and S-UNIWARD. For this setup, the enrichment of the learning set is not strongly unbalanced (1 BOSS pair for 2 LIRMM pairs), done with images acquired with different cameras but processed with the same development. **It seems that for a beneficial enrichment, the additional images have to be acquired with the same cameras**. Additional facts seem to confirm this hypothesis in Section 7.3.3 and Section 7.3.4.

When enriching the *BOSSBase* with *BOSS + LIRMM + BOWS2*, the results are as good (or a slightly better for WOW) as using the *BOSSBase* alone. Finally, the results become better when *BOSS+BOWS2+LIRMM2+VA* is used, but the increase in performance is only of 0.9% for S-UNIWARD (resp. 2% for WOW), while using the *BOSS+BOWS2* (see Table. 7.1) give 2% increasing for S-UNIWARD (resp. 4% for WOW).

Those results confirm again that performance is increased if there is an enrichment with images acquired with the same cameras and with the same development (BOWS-2 share similar cameras and a similar development). This tendency seems to contradict the idea that using millions of images, whose distribution is diverse, would be the best solution for increasing the steganalysis results [Zeng et al.,

2017b]. Indeed, the added images have to share a very similar "distribution", and images have probably to be acquired with the same cameras. In Section 7.3.3 we explore a little bit more this hypothesis.

### 7.3.3 Setup 3: Enrichment with strongly dissimilar sources and unbalance proportions

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.7** % |
| BOSS+PLACES2 1% | 34.2 % | 41.6 % |
| BOSS+PLACES2 10% | 40.0 % | 43.9 % |
| BOSS+PLACES2 100% | 44.6 % | 45.3 % |

TABLE 7.3: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with a learning base augmented with different portions of PLACES2. .

In Table. 7.3, we report the results with *no enrichment* (noted **BOSS**), the results with 1% of PLACES2 enrichment (noted **BOSS + PLACES2 1%**), the results with 10% of PLACES2 enrichment (noted **BOSS + PLACES2 10%**), and 100% of PLACES2 enrichment (noted **BOSS + PLACES2 100%**). Note that for *PLACES2 1%*, the training set is made of 14 000 pairs (without counting the validation), for *PLACES2 10%*, the training set is made of 104 000 pairs (without counting the validation), and for the *PLACES2 100%*, the training set is made of 1 004 000 pairs (without counting the validation).

Whatever the enrichment and whatever the embedding algorithm, the results are always worse than when using the BOSSBase alone. Respectively when 1%, 10% and 100% of the images in the PLACES2 database are added to the learning, the results get worse and worse, with respectively an increase of the detection error for S-UNIWARD (resp. WOW) of 5% (resp. 6%), 7% (resp. 12%), and then 9% (resp. 17%). Note that with an enrichment containing all images of PLACES2 (1 BOSS pair for 251 PLACES2 pairs), the detection is close to a random guessing.

Since the distribution of BOSS and PLACES2 are totally different (PLACES2 results from a JPEG dequantization, and a very diverse set of sources of cameras), the BOSS distribution is lost, and since no re-balancing measures are used during the learning, the BOSS distribution is considered as anecdotal and it is not really taken into account during the learning. Practically, the total loss computed for BOSS images is negligible compared to the total loss computed for PLACES2 images, and thus a minimization of the global loss will mainly concentrate on minimizing the loss associated to the PLACES2 images. Coming back to our previous statement, **using millions of images is not sufficient** [**Zeng et al., 2017b**]**, the added images have to share a very similar "distribution", and images have probably to be acquired with the same cameras**.

### 7.3.4 Setup 4: Enrichment with the same RAW images but with a different development

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.7** % |
| BOSS+DEV:Res-Bicub | 25.7 % | 37.5 % |
| BOSS+DEV:Res-Spline | 26 % | 35.8 % |
| BOSS+DEV:Res-NoInt | 25.6 % | 36.2 % |
| BOSS+DEV:Crop | 34.8 % | 44.2 % |
| BOSS+DEV:Res-Crop | 28.1 % | 37.9 % |
| BOSS+BOSS-ALP | 26.0 % | 35.5% |

TABLE 7.4: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with a learning base augmented with different BOSSBase versions..

In Table. 7.3, we report the results with *no enrichment* (noted **BOSS**), and the results with 6 different versions of the BOSSBase, each generated from the RAW images. There are enrichment using a resizing performed respectively with a *bicubic interpolation* (noted **BOSS+DEV:Res-Bicub**), a *spline interpolation* (noted

**BOSS+DEV:Res-Spline**), one *without any interpolation* (noted **BOSS+DEV:Res-NoInt**), one without resizing but with a *central crop* (noted **BOSS+DEV:Crop**), one with a resizing to a 768×768 images *without any interpolation* and then a *central crop* (noted **BOSS+DEV:Res+Crop**), and finally an enrichment with the use of Adobe Photoshop Lightroom 6 instead of *ImageMagick*, for generating the color images and then resizing to $256 \times 256$ the images while keeping the width/length ratio (noted **BOSS-APL**).

From Table. 7.4, we can observe that the enrichment with a *crop development* (BOSS+DEV:Crop) lead to very bad results. The increase of the detection error of 7% for S-UNIWARD (resp. 7% for WOW). The enrichment with a resize to 768×768 followed by a crop (BOSS+DEV:Res+Crop), to a lesser extent, also give bad results with an increase of the detection error of 1% for S-UNIWARD (resp. 0.3% for WOW). Those bad results suggest that a resolution change during the development has a strong impact on the pixels distributions. When looking to the extreme case of the *crop development* (BOSS+DEV:Crop), we easily understand that the resulting images content change; there is almost no variations and no edges. Thus, **an enrichment with a BOSS version whose development does not ensure the same final pixel resolution than BOSS Base will not enrich favourably the learning data-base**.

In counterpart, using the same resize procedure with a slight variation on the *interpolation* (spline, no-interpolation, bicubic), or with the *Adobe Photoshop Lightroom Process* allows scrounging at most 1% for S-UNIWARD (resp. 2% for WOW). This confirms that additional samples very close to the target BOSS distribution can improve the learning capabilities. **Looking back to the various experiment done previously, one can observe that in order to enrich favourably a target database, a useful "trick" is to use images acquired with the same cameras than the target database, and to use a very close resizing process than the one used for the target database.**

In order to push the reflection a little bit more, we made an additional experiment where we regrouped diverse versions of BOSSBase (BOSS+DEV:Res+Bicub,

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.7** % |
| BOSS+all-DEV | 23.0 % | 33.2 % |
| BOSS+BOWS2 | 23.7 % | 34.4% |

TABLE 7.5: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with a learning base augmented with different versions of BOSSBase..

BOSS+DEV:Res+Spline, BOSS+DEV:Res+NoInt, BOSS+DEV:Res+Crop) to the exception of BOSS+DEV:Crop. In Table. 7.5, we report the results with this gathering of various development (noted **BOSS+all-DEV**), and the results with LIRMM and BOWS2 enrichment (noted **LIRMM+BOWS2** and already reported in Section 7.3.1). Note that for *BOSS+all-DEV*, the training set is made of 44 000 pairs (without counting the validation), and for *LIRMM+BOWS2* the training set is made of 14 000 pairs (without counting the validation).

For those two enrichments, there is a real improvement with a decrease of the error probability of detection of 2-3% for S-UNIWARD (and 4% for WOW). This last result is very interesting and shows that in order to enrich a database, in a practical scenario, there are at least those two options:

Given a target database:

- either Eve (the steganalyst) finds similar cameras (used for generating the target database), capture new images, and reproduce the same development than the target database, with a special caution to the resizing,
- either Eve has an access to the original RAW images and reproduces similar developments than the target database with the similar resizing,

The reader should also remember that Virtual Augmentation is also a good cheap processing measure.

Note that it is unclear which option would be better in a practical case. Additional experiments have to be done in the future. Anyway, those two enrichments show that a very caution process has to be taken for really improving the results.

We believe that those enrichments reduce the over-fitting and also improve the generalization of the learner.

### 7.3.5 Setup 5: Enrichment with a re-processing of the learning set

In all previous setups, given a target database (never compressed 8-bits grey-level 256×256 images), we were presuming either a prior knowledge of the cameras used for the images acquisitions or a direct access to the RAW versions of the original images. In real-world cases, those knowledge are most of the time not available. Moreover, retrieving the camera models is a very complicated task in a real scenario due to the huge number of cameras.

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.7** % |
| BOSS+DEV:Translation | 34.7.0 % | 47.8 % |
| BOSS+DEV:Up-Down-Sampling | 31.2 % | 42.6 % |
| BOSS+DEV:pixels-off (d=100) | 25.1 % | 35.6 % |

TABLE 7.6: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with a learning base augmented with a re-processing of BOSSBase..

In Table. 7.6, we report the results with *no enrichment* (noted **BOSS**), and the results with 3 different reprocessed versions of the BOSSBase, each generated from the original 256×256 8-bits grey-level BOSSBase images.

The first reprocessing (noted **BOSS+DEV:Translation**) consists in applying a sub-pixel image translation, of 0.5 pixels, on the padded (symmetric padding) images, and then applying a crop operation to re-obtain a 256×256 images.

The second reprocessing (noted **BOSS+DEV:Up-Down-Sampling**) consists in applying a Lanczos3 filter for the up-sampling in order to obtain a 512×512 images,

and then down-sampling with the same interpolation Kernel to re-obtain images of 256×256 size.

The third reprocessing (refereed as **BOSS+DEV:pixels-off**) consists of selecting a random number $d$ of the image pixels and setting them to 0. Different values of the portions $d$ are tested($d = 100, d = 200, d = 400$ and $d = 1024$). Once the database of images with off-pixels is generated, an embedding process is launched to obtain the stego images.

For the two first reprocessing, the results are catastrophic, with an increase of the error probability of 6% to 11% for S-UNIWARD and 4% to 7% for WOW. However, as for "off-pixel" reprocessing great results are obtained. The error probability is decreased by 1% to 2% for WOW and by more than 1% to % S-UNIWARD.

Since BOSS+DEV:pixels-off represents an easy development that obtains good results, it was intuitive to investigate its true potential. To this end, we have made some several extra experiments. Firstly, we try to use not only one reprocessed base but two. Thus, we generate another redeveloped base and this time $d$ is set to 200. Thus, the learning set is composed of 24000 pairs (cover/stego). This setup is referred to as BOSS+DEV:pixels-off-1. Secondly, we make BOSS+DEV:pixels-off-1 learning set even bigger by adding another processed base obtained by turning off more pixel $d = 400$. This setup is named as BOSS+DEV:pixels-off-2. The last setup is called as BOSS+DEV:pixels-off-3. In this setup, we take the learning set of BOSS+DEV:pixels-off-2 setup and we add a third processed base. This latter is obtained by setting $d = 1024$. The results are given below.

From Table. 7.7, we can observe that the enrichment with *two reprocessed bases* (BOSS+DEV:pixels-off-1 (d=100, d=200)) improves even more the results compared to the BOSS+DEV:pixels-off (d=100). The decrease of the detection error is of 1% for S-UNIWARD (resp. almost 2% for WOW). The enrichment with *three processed bases* (BOSS+DEV:pixels-off-2 (d=100, d=200, d=400)), to an extent, also give good results with a decrease of the detection error of 1% for S-UNIWARD (resp. 0.7% for WOW). When looking to the last setup (BOSS+DEV:pixels-off-3

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.7** % |
| BOSS+DEV:pixels-off (d=100) | 25.1 % | 35.6 % |
| BOSS+DEV:pixels-off-1 (d=100, d=200) | 24.6 % | 34.5 % |
| BOSS+DEV:pixels-off-2 (d=100, d=200, d=400) | 24.1 % | 34.1 % |
| BOSS+DEV:pixels-off-3 (d=100, d=200, d=400, d=1024) | 24.7 % | 34.2 % |

TABLE 7.7: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with a learning base augmented with pixels-off re-development of BOSSBase ..

(d=100, d=200, d=400, d=1024), the results get worse, with respectively an increase of the detection error for S-UNIWARD (resp. WOW) of 0.1% (resp. 0.6%).

We easily understand that this type of reprocessing the learning set (disabling some pixels) is beneficial to improve detection accuracy, but disabling to many pixels can have the contrary effect due to the change in data distribution ((BOSS+DEV:pixels-off-3 (d=100, d=200, d=400, d=1024), thus the accuracy is decreased.

Those results show that we can directly use the original base to augment the learning set; thus it may be pointless to spend much time looking for the RAW images or to spend money buying the same cameras.

#### 7.3.5.1 Extra: Different development using a totally different program

Although we have experimented different developments in the previous subsection, we were using the same program for all developments. In order to exclude the hypothesis that using the same program may influence the images distribution, we decided to conduct an extra experiment by using a different program. For that and in this subsection, we use Adobe Photoshop Lightroom 6.

With the help of this program, two versions of BossBase V1.01 have been created. The first version is noted BossBase 2.0.1, and the second one BossBase 2.0.2.

- BossBase 2.0.1 version is similar to the original BossBase since we keep the same configuration. Like the original BossBase v1.01, we use the original RAW images to generate a colour image with a size of 256 by using a resize function followed by a crop. The last step we use Matlab to obtain "PGM" images.

- BossBase 2.0.2 is Generated the same way as BossBase 2.0.1 but with an extra manipulation. We decreased the exposure in all image by shifting the entire histogram to the left, to highlight more the shadows and the edges and slightly change the images distribution.

| | WOW 0.2 bpp | S-UNIWARD 0.2 bpp |
|---|---|---|
| BOSS | **27.8** % | **36.7** % |
| BOSS+BOSS 2.0.1 | 26.0 % | 35.5% |
| BOSS+BOSS 2.0.2 | 27.43 % | 37.0% |

TABLE 7.8: Base Augmentation influence: error probability of Yedroudj-Net, on WOW and S-UNIWARD at 0.2 bpp with a learning base augmented with two different versions of BOSSBase. .

From Table. 7.8, we can observe that using another program does not have that big influence on the learning set distribution, thus the obtained results over the "BOSS+BOSS 2.0.1" setup are rather positive. A decrease of the prediction error of 1.8% (resp. 1.2%) for WOW (resp. S-UNIWARD) is achieved. As for "BOSS+BOSS 2.0.2" we obtain the same results as on the no enrichment setup (BOSS).

## 7.4 Conclusion

In this chapter, we have explored ways to enrich a learning database when steganalysis is done with a CNN. The enrichment is a crucial task since, in the majority of the today's experiments, the required number of images have to be extremely high due to the huge number of parameters to be learned. Using an insufficient

set of examples (images) leads to CNNs that have not "learned enough" and the average efficiency is thus reduced.

After recalling the state-of-the-art of 2017 for the spatial CNN steganalysis, and briefly recalling the state-of-the-art steganalysis approach named Yedroudj-Net, we have presented various results. Additionally, to the classical data augmentation, which consists of applying flips and rotations on the learning images [Krizhevsky et al., 2012], we observed three other ways for favourably enriching the learning database. The trend is that, in a clairvoyant scenario (knowledge of the embedding algorithm, knowledge of the payload size, approximate knowledge of the of the images distribution), for a given target (test) database, in order to augment its learning database, the steganalyst (Eve) has three choices:

- Either she is able to guess the camera used for generating the target database. She thus captures new images, and reproduce a similar development than the target database, with a special caution to the resizing,
- Either she has access to the original RAW images and reproduces a similar development than the target database with the similar resizing.
- Either she clones the database disable a very small percentage of images pixels (pixels-off).

The two first ways to enrich the database are very restrictive. The last way is preferred as its an easier way and more practice in a real-world situation. As explained previously, some complementary solutions can be used such as transfer learning [Qian et al., 2016], or the use of ensembles [Xu et al., 2016b], but the underlying questions of generalizations / cover-source mismatch have to be explored deeper in the future.

# Chapter 8

# Steganography using a 3 player game

## Contents

## 8.1 Introduction

Image steganography aims to securely embed secret information into cover images. Until now, adaptive embedding algorithms such as S-UNIWARD or Mi-POD, were among the most secure and most used methods for image steganography. However, these algorithms are considered as *naive adaptive steganography*. They are indeed designed without any consideration of how the steganalyst (Eve) is evolving when it is confronted to this embedding.

With the emergence of deep learning techniques and more specifically Adversarial Generative Networks (GANs), the design of *strategic adaptive steganography* models has become much easier, leading to the appearance of new steganographic techniques. Among them, there is the 3-player game approach, where three networks compete against each other. Those networks are refereed as *Agent-Alice* for the embedding network, *Agent-Bob* for the extracting network, and *Agent-Eve* the steganalysis network. See Section. 4.4

In this chapter, we present our proposed steganographic system, which is based on the 3-player game approach. Three different architectures are developed. The first architecture is proposed as a rigorous alternative to two recent publications which represent the state-of-the-art of the 3-player game approach [Hayes and Danezis, 2017, Zhu et al., 2018]. The second stands for an improvement of the first one where the stego noise power is better handled. Finally, our third architecture enriches the second one with a better interaction between the embedding and extracting networks. Our method achieves better results compared to the existing works and paves the way for future researches.

In this chapter, Section. 8.2 focuses on the steganography's main concept with *3-player game*. In Section. 8.3, we recall the propositions given in [Hayes and Danezis, 2017] and [Zhu et al., 2018]. In Section. 8.4 we present three architectures in order to resolve previous unsolved problems. In Section. 8.5 we give some experimental results and their analysis. Finally, we conclude in Section. 8.6.

## 8.2 General concept

Let $\mathbf{x} \in \{0, ..., 255\}^{w \times h}$ be a cover matrix composed of $w \times h$ pixels, and $\mathbf{y} \in \{0, ..., 255\}^{w \times h}$ be a stego matrix with a size of $w \times h$ pixels generated by the steganographic system more precisely the **Agent-Alice**. We note **m** a secret binary message vector of $m$ bits that **Agent-Alice** wants to send to **Agent-Bob**, and **m'** the binary message extracted by **Agent-Bob**, where **m'** has the same length as **m**. Let **k** be the shared key between **Agent-Alice** and **Agent-Bob,** where **k** is a $k$-sized binary vector. Let us further note $\mathbf{i} \in \{0, ..., 255\}^{w \times h}$ an image with an unknown label. We use the notation $y$ for the image label where $y \in \{0,1\}$, $y$=0 if **i** is a cover, and $y$ =1 if **i** is a stego.

Bearing in mind that the objective of the steganographic system based *3-player game*, illustrated in Figure. 4.6, is to learn a model so that Agent-Alice can generate a stego **y** by embedding the secret message **m** within the cover **x**, and then secretly communicate it to Agent-Bob (as defined in **Algorithm 1**), a loss function is given to each agent:

**Agent-Eve's loss:** Agent-Eve is modelled by a function *Agent-Eve*: $\mathbf{i} \rightarrow [0, 1]$, which takes the image **i** and returns a real score between 0 and 1, such that 0 corresponds to a cover and 1 corresponds to a stego.

Agent-Eve's general loss consists then in minimizing the distance between the label $y$ and Agent-Eve's prediction:

$$\mathcal{L}_{Eve} = dist(y - \textit{Agent-Eve}(\mathbf{i})). \tag{8.1}$$

The distance used for Agent-Eve's loss is usually the cross-entropy distance; thus the loss of Equation. 8.1 is given as:

$$\mathcal{L}_{Eve} = - \, y \cdot log(\textit{Agent-Eve}(\mathbf{i})) - (1 \text{ - } l) \cdot log(1 - \textit{Agent-Eve}(\mathbf{i})). \tag{8.2}$$

**Agent-Bob's loss:** Agent-Bob attempts to reconstruct the secret message **m** from the received image **y** using the key **k**. The reconstructed message **m'** should be equal to **m** (**m** = **m'**). To this end, Agent-Bob's loss consists in minimizing a distance between **m'** and **m** (usually a L2 distance):

$$\mathcal{L}_{Bob} = dist(\mathbf{m}, \mathbf{m'}). \tag{8.3}$$

**Agent-Alice's loss:** Agent-Alice objectives are multiple. The first is to generate a stego image **y** that is close enough to the cover **x**. The second is to allow Agent-Bob to reconstruct the secret message **m** correctly from the stego image. The third is that Agent-Eve accuracy should be not better than a random guess whether a given image **i** is a cover or stego (50-50 chance of making the good guess). The loss of Agent-Alice is then the weighted sum of three terms: $\mathcal{L}_{bob}$, $\mathcal{L}_{Eve}$, and $dist(\mathbf{x},\mathbf{y})$ the distance calculated between **x** and **y**, where the coefficients used for the weighted sum, $\lambda_A$, $\lambda_B$, $\lambda_E$, belongs to [0,1] and their sum is equal to one in order to adjust the contribution of each term to the loss of Agent-Alice:

$$\mathcal{L}_{Alice} = \lambda_A \cdot dist(\mathbf{x},\mathbf{y}) + \lambda_B \cdot \mathcal{L}_{Bob} - \lambda_E \cdot \mathcal{L}_{Eve}. \tag{8.4}$$

Note that pixels values from **x** and **y** are all normalized by a division by 255. Each of the three terms has thus, similar values ranges, which is a practical requirement in an optimization process (see Figure. 8.6).

The training process of the steganographic system based 3-player games is described in **Algorithm 1**.

Before proceeding with the presentation of our proposed 3-player steganographic system, we start by presenting previous works that do belong to the same steganography GAN family, (i.e. *3-player game* family).

## 8.3 Related work

In this subsection, we recall the architectures of GSIVAT [Hayes and Danezis, 2017] and HiDDen [Zhu et al., 2018]. These two architectures were proposed basically for steganography purposes.

### 8.3.1 Generating Steganographic Images Via Adversarial Training (GSIVAT)

In [Hayes and Danezis, 2017], the authors propose a steganographic system (GSIVAT) composed of three neural networks, each one representing one agent (Agent-Alice, Agent-Bob, Agent-Eve). We sum up the GSIVAT architecture in Figure. 8.1.

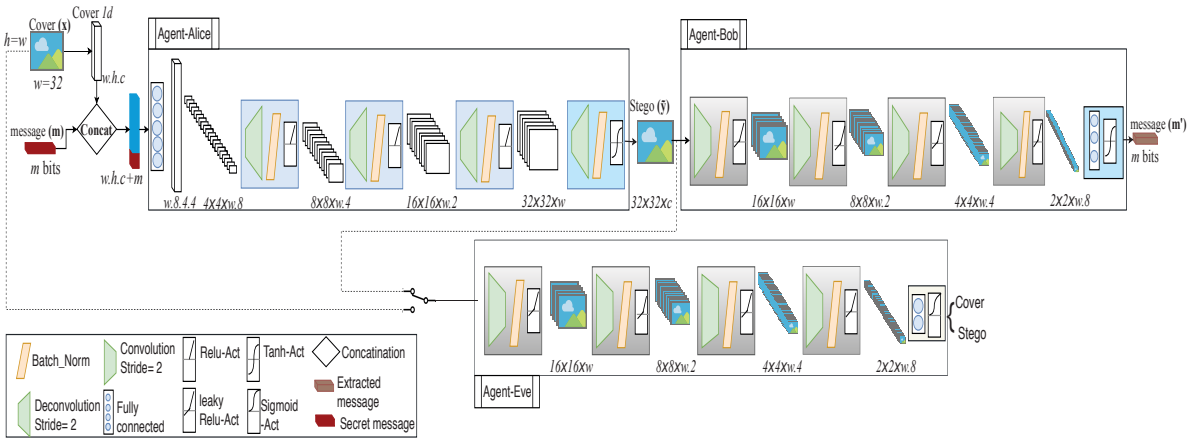



FIGURE 8.1: GSIVAT [Hayes and Danezis, 2017] architecture.

Their system's input are a cover image $\mathbf{x}$, a *3D* vector, whose size is $w \times h \times c$ (where $c$ is the channel number) and a secret message $\mathbf{m}$ of $m$ bits. $\mathbf{x}$ is flattened to a 1D vector and concatenated with $\mathbf{m}$, the resulting vector size is $w \cdot h \cdot c + m$. This vector is then fed to Agent-Alice.

**Agent-Alice** is composed of one fully connected layer (FC) and four deConv-BN-Act blocks (see Figure. 8.1). Each block is the combination of a deconvolution

(deConv), a batch normalization (BN) and one activation function (Act) layer (ReLU, leaky-ReLU, sigmoid, tanh).

The number of neurons in the fully connected (FC) layer is the multiplication result of $w{\cdot}8{\cdot}4{\cdot}4$. The output of the FC is reshaped to $w{\cdot}8$ of $4{\times}4$ sized features-maps. Then, these feature-maps fed four *deConv-BN-Act blocks*. After each block, the size of these features-maps is up-sampled by a factor of two, and the channel number is reduced by a factor of two. In the end, an image of the same size as $\mathbf{x}$ is obtained.

Agent-Alice outputs the stego image $\tilde{\mathbf{y}} \in \mathbb{R}^{w \times h}$ which is fed to both Agent-Bob and Agent-Eve. The distance between $\mathbf{x}$ and $\tilde{\mathbf{y}}$ (dist($\mathbf{x}$,$\mathbf{y}$) in Equation. 8.4) is the L2 distance.

**Agent-Bob**'s architecture is depicted in Figure. 8.1. It consists of four *Conv-BN-Act* blocks and one FC layer. Each block is made of a fractionally-strided (stride=2) convolution layer (Conv), a batch normalization layer and an activation function "Leaky ReLU" [Maas et al., 2013]. The FC layer is made of $m$ neurons followed by a tanh activation function.

Agent-Bob outputs the extracted message **m'**. The L2 distance is used to compute the loss previously given in Equation. 8.3.

**Agent-Eve** takes as input image **i**, and outputs a score determining the label of the input (cover/stego). The architectures of Agent-Eve and Agent-Bob are similar except for the fully connected layer having only two neurons, and the activation function replaced by a sigmoid. The sigmoid cross entropy loss is used for Eve's loss (see Equation. 8.2).

### 8.3.2 HiDDeN: Hiding Data With Deep Network

In [Zhu et al., 2018], the authors propose a model for image steganography, but also for watermarking (called HiDDen). We will only discuss the architecture of their steganographic system.

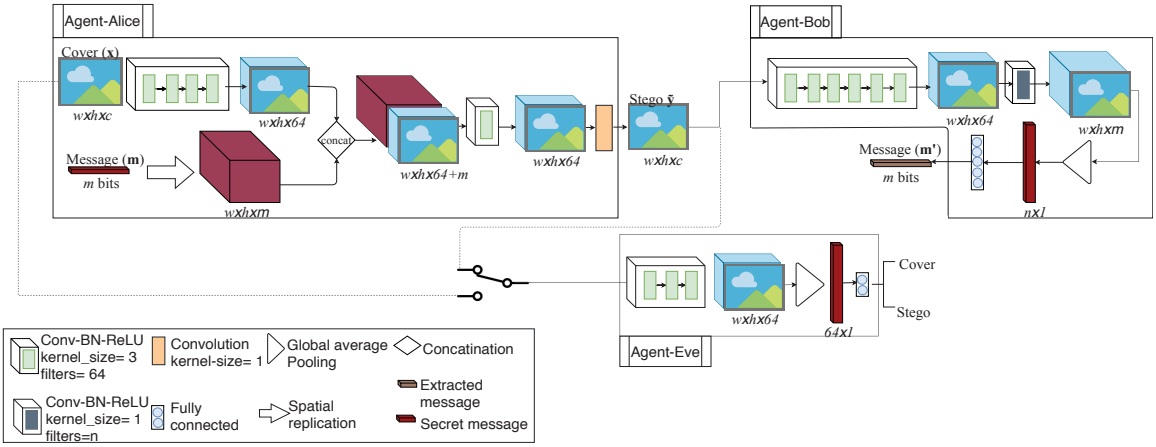



FIGURE 8.2: HiDDeN architecture.

Their system takes as input a $w \times h \times c$-sized cover image $\mathbf{x}$ and a $m$-length secret message $\mathbf{m}$. As all *3-player game* steganographic systems, their proposed system is composed of three neural networks. The structure of these three networks is generally composed of blocks called *Conv-BN-ReLU* which stands for the combination of Convolution, Batch Normalization and ReLU activation function (see Figure. 8.2).

**Agent-Alice** is composed of five *Conv-BN-ReLU* blocks and one convolution layer with a kernel size of 1×1. Firstly, the cover image $\mathbf{x}$ goes through four *Conv-BN-ReLU* blocks to obtain an intermediate representation image $\tilde{\mathbf{x}}$. On the other hand, the message $\mathbf{m}$ is replicated so that the resulting size is $w \times h \times m$. Secondly, $\tilde{\mathbf{x}}$ is concatenated with the replicated message and fed to another *Conv-BN-ReLU* block with 64 output filters. A final convolution layer with a 1×1 kernel is used to generate the stego image $\tilde{\mathbf{y}} \in \mathbb{R}^{w \times h}$. The loss of Agent-Alice is calculated using the loss of Equation. 8.4.

**Agent-Bob** is composed of seven *Conv-BN-ReLU* blocks, followed by a global spatial average pooling (to produce a vector with the same size as the message). Then, a single, fully connected layer ends the architecture, as shown in Figure. 8.2. Agent-Bob receives the stego image $\tilde{\mathbf{y}}$, and produces the predicted message **m'**. The distance used in Agent-Bob's loss of Equation. 8.3 is the l2 distance.

**Agent-Eve** takes an image as input, and outputs a score indicating whether the given image is a cover or stego one. Agent-Eve has an architecture similar to

Agent-Bob, but only has three *Conv-BN-ReLU* blocks instead of seven. The last layer is a FC layer with an output size of two units (see Figure. 8.2). The authors adopt the use of the cross-entropy loss presented in Equation. 8.2 for Agent-Eve.

### 8.3.3 Discussion

Although GSIVAT [Hayes and Danezis, 2017], and HiDDen [Zhu et al., 2018]) offer some interesting ideas, they both have serious flaws:

Firstly, neither of the two approaches use a shared secret key during the embedding/extracting process. The authors of [Hayes and Danezis, 2017] and [Zhu et al., 2018] suppose that the information about model weights, architecture, and the set of images used for training is shared between Agent-Alice and Agent-Bob. According to the authors, this shared information can be considered as the secret key. Besides the fact that such a hypothesis is heavy in size (almost 70 Mb need to be sent to Agent-Bob [Hayes and Danezis, 2017]), it is also the equivalent of performing an embedding process with the same *secret-key*. Such embedding process is highly not recommended in steganography as it leads to a very strong detectability [Pibre et al., 2016].

Secondly, there is no discretization module for the generated images (Agent-Alice provides $\tilde{\mathbf{y}}$ instead of $\mathbf{y}$). In a real-world situation [Ker et al., 2013], Bob receives an image whose values are defined in {1,..., 255}, and has to extract the secret message. In [Hayes and Danezis, 2017] and [Zhu et al., 2018], Agent-Alice generates real-valued images i.e. **not** discrete-values images, and these images are fed to Agent-Bob. This makes both Agent-Alice and Agent-Bob useless in practice. Alice needs to provide Bob with images that are not suspicious, meaning images with discrete values. Indeed, images have to be formatted in pgm, or any lossless compressed images format. Note that if Alice decides to round the real-values images (generated by the Agent-Alice) in order to discretize them in {1,..., 255},

Bob, when using Agent-Bob algorithm, will not extract correctly the message since Agent-Bob has been built for real-values images[1].

Thirdly, the computation load is a serious issue that must be taken into account when working with deep learning. GSIVAT authors [Hayes and Danezis, 2017] have worked on 32×32-sized images while Hidden authors [Zhu et al., 2018] have used 16×16-sized patches. This limitation for the size of the images is due to the use of the FC layers, which induce expensive memory and computation costs. The authors justify that working on large images could be completed by treating bigger images with a separate treatment for each part of the image. This is indeed a bad idea since statistical traces may be found at the block boundaries and would lead to an easily detectable embedding scheme (See for example the discussion in Section.4.2 of [Fridrich, 2009], or the dependencies preservation between blocks in JPEG steganography [Taburet et al., 2019]).

Finally, note that in these two papers, the experimental steganalysis is performed with the algorithm ATS [Lerch-Hostalot and Megìas, 2016] proposed in 2015. This algorithm is basically proposed to handle the cover-source mismatch problem (seen in Section. 5.3.3), which is definitely not the appropriate scenario to evaluate the empirical security of an embedding algorithm (especially when it is a *strategic embedding* algorithm). Indeed, ATS is based on the assumption of *constant noise direction* in the embedding space, which may not be true for *a strategic adaptive* algorithms. The empirical security is probably undervalued when compared to an Ensemble Classifier/Rich Model (EC+RM) [Kodovský et al., 2012, Fridrich and Kodovský, 2012], Yedroudj- Net [Yedroudj et al., 2018b], ReST-Net [Li et al., 2018], or SRNet [Boroumand et al., 2019]. In addition, these four steganalysis algorithms represent the state-of-the-art of steganalysis, so their use makes more sense.

Considering the errors made in previous works based *3-player game*, we proposed a new system which, on the one hand, corrects the errors of previous work and,

[1]We observed this phenomenon during our experiments.

on the other hand, offers a good level of security and correctly carry out the embedding/extraction processes.

## 8.4 Our steganographic system's Architecture

In this section, we present our new *strategic adaptive steganography* system based on the *3-player game* concept. We are obtaining an embedding algorithm (Agent-Alice) and an extracting algorithm (Agent-Bob) functional in practice. We therefore:

1. Integrate a stego-key for the input of Agent-Alice and Agent-Bob. With two different stego-keys, Agent-Alice will generate two different stego images. Alice knows that she must change its stego-key very often if she does not want to be caught [Pibre et al., 2016]. By extension, knowing that it is easier to break a system that always uses the same key, it is important to integrate a stego key in the input of Agent-Alice and Agent-Bob in order to avoid the counter-productivity that a unique key could have on the convergence of Agent-Alice and Agent-Bob facing Agent-Eve. This argument is not considered at all in [Hayes and Danezis, 2017] and [Zhu et al., 2018] and can be a major flaw in their performances;

2. handle the problem of discretization in order for Alice to be able to send to Bob, through e-mail, memory card, cloud storage, an image in a non-suspicious standard format. [Hayes and Danezis, 2017] and [Zhu et al., 2018] do not deal with this fundamental issue;

3. guarantee a scalable (in memory and in computation) solution thanks to an architecture that consists in only convolutions. Thus, it can deal with image dimensions usually used in deep-learning and steganalysis by deep-learning in the academic experiments ($255 \times 255$ or $512 \times 512$) [Chaumont, 2020]. The convolutional architectures also allow deeper networks to deal with harder problems modelization. GSIVAT [Hayes and Danezis, 2017]

works with $32 \times 32$ images and HiDDeN [Zhu et al., 2018] works with $16 \times 16$ images and they both use very small networks.

Additionally, our proposed method offers two interesting properties. First, it is "bit-rate adaptive". Indeed, there is no need to re-train the system each time we change the bit rate, i.e. each time the message size is different (this is not the case for [Hayes and Danezis, 2017] and [Zhu et al., 2018]). Secondly, we adopt a strong steganalyst for Agent-Eve, which at the most ensures better security, and at less is an up-to-date steganalysis.

We propose three different architectures for our steganographic system. These architectures illustrate three different solutions going from the basic one to the more adapted solution. On these three architectures, Agent-Eve's architecture remains the same, while Agent-Alice's and Agent-Bob's architecture change their designs. The first architecture is presented to illustrate the use of a secret shared key during the embedding. The second architecture intends to reduce the power of noise introduced by embedding the hidden message into the cover image. Finally, the third architecture tries to improve the performances of message extraction while linking Agent-Alice and Agent-Bob's behaviour.

### 8.4.1 The training process

Over the three proposed architecture, the training procedure of the system remains the same. We alternate the training between the three agents Agent-Alice, Agent-Bob and Agent-Eve, where Agent-Alice and Agent-Bob are jointly trained as a single network, and Agent-Eve is trained separately.

First, Agent-Bob and Agent-Alice are trained on a fixed number of mini-batch using the two models of Agent-Alice and Agent-Bob saved previously. See Algorithm. 1. This training process is repeated for several loops until all losses tend to be constant.

### 8.4.2 The proposed architecture of the Agent-Eve

Agent-Eve tries to guide both Agent-Alice and Agent-Bob through the process of learning a *strategic adaptive embedding* algorithm. If Agent-Eve is weak, the 3-agent system falls down because Agent-Alice and Agent-Bob will no longer search for better solutions as Agent-Eve cannot cope with their evolution. To this end, it is essential to adopt a strong steganalyzer.

In 2018, the best spatial steganalyst was, Yedroudj-Net [Yedroudj et al., 2018b,a] (see Section. 6.2), ResT-Net [Li et al., 2018] (see Section. 5.5.3) and very recently SRNet [Boroumand et al., 2019] (published in September 2018). Among these networks, Yedroudj-Net is the shallowest network with six convolution layers compared to 25 layers for SRNet, and 30 layers for ReST-Net (3 sub-networks each containing 10 layers). Besides its affordable size, Yedroudj-Net training does not require the use of any tricks that could increase the computational time. This network is therefore well adapted to Agent-Eve, especially knowing that *the 3-player approach* takes a lot of time before it converges to a good solution. Additionally, Yedroudj-Net can easily be improved in the future if required [Zhang et al., 2019].

Hence, Agent-Eve architecture is the same as the Yedroudj-Net presented in Figure. 6.1. It is composed of 7 blocks: a pre-processing block, five convolutional blocks, and a fully connected block made of three fully connected layers followed by a softmax. Agent-Eve's network is trained by minimizing the loss given in Equation. 8.2.

### 8.4.3 First-Architecture

The first architecture is similar in spirit to the previous approaches that exist in the literature.

Agent-Alice's network receives a $m$-length secret message $\mathbf{m}$, a key $\mathbf{k}$ of $k$ bits, and a cover $\mathbf{x}$. In order to concatenate the cover $\mathbf{x}$ with the message $\mathbf{m}$, both should have the same size. We therefore use the key $\mathbf{k}$ to spread out the secret message

$\mathbf{m}$ in a matrix noted as $\mathbf{s^{(m)}} \in \{0,1\}^{w \times h}$ that is initialized with zeros and has the same size than our cover image.

The spreading of $\mathbf{m}$ in $\mathbf{s^{(m)}}$ is obtained by using a pseudo random number generator (PRNG) seeded by the key $\mathbf{k}$. The PRNG sequentially pick a bit of $\mathbf{m}$ and fills a *non-used* position in the $\mathbf{s^{(m)}}$ matrix. The filled positions define the binary mask $\mathbf{\Omega} \in \{0,1\}^{w \times h}$. $\mathbf{\Omega}$ thus contains exactly $m$ ones.

Note that with the knowledge of $\mathbf{k}$, and the index of a bit $m_i$ in our message $\mathbf{m}$ with $i \in \{1, ..., \mathrm{m}\}$, we can deduce the position $(u, v) \in \{0, ..w\} \times \{0, ..h\}$ where this bit is stored in $\mathbf{s^{(m)}}$, and inversely from a position $(u, v) \in \{0, ..w\} \times \{0, ..h\}$, we can deduce the bit $m_i$ with $i \in \{1, ..., \mathrm{m}\}$ of the message $\mathbf{m}$.

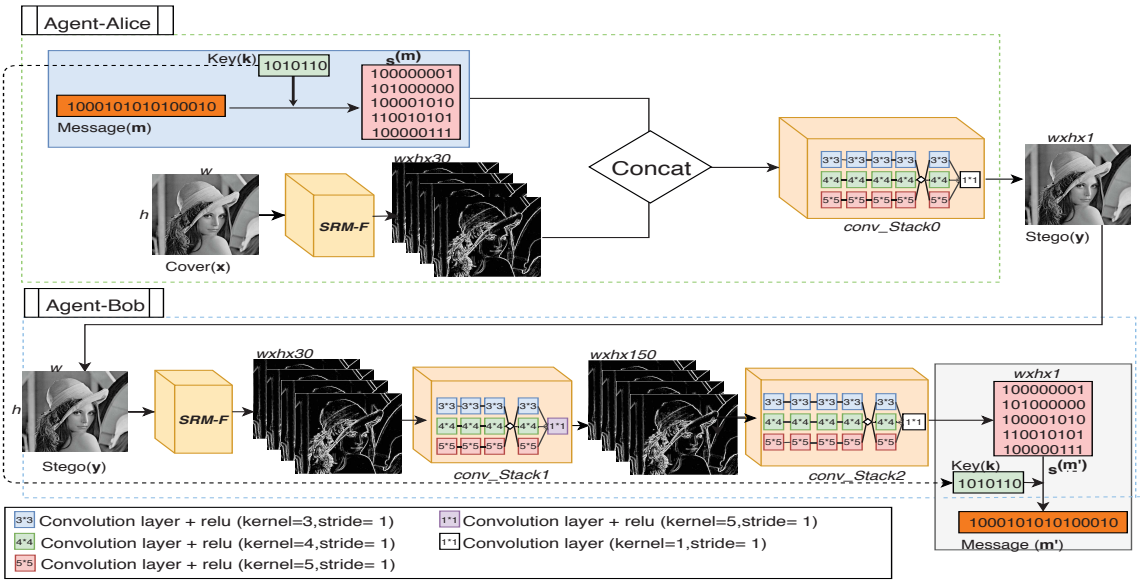



FIGURE 8.3: Agent-Alice and Agent-Bob with the first architecture.

The cover image $\mathbf{x}$ is fed to a convolution layer called *SRM-F*. Its weights are initialized with the 30-basic high-pass filters of SRM [Fridrich and Kodovský, 2012], similarly to Yedroudj-Net [Yedroudj et al., 2018b]. The output is then concatenated with $\mathbf{s^{(m)}}$ and fed to *conv_Stack0* which is composed of a set of convolution layers. The output is a $w \times h$ image which represents the stego image $\mathbf{y}$ (see Figure. 8.3).

Agent-Bob uses the stego $\mathbf{y}$ and the key $\mathbf{k}$ to output the predicted message $\mathbf{m'}$, a vector of $m$ bits. First, the stego image passes through a convolution layer similar to *SRM-F*. Next, the obtained feature-maps from *SRM-F* goes through

another set of convolutions in two *conv_Stack1, conv_Stack2*. Note that *conv_-Stack2* share the same construction with *conv_Stack0*, but not the same number of features-maps for each layer. The extracted message **m'** is then obtained (see Figure. 8.3).

The back-propagation is done by minimizing both losses of Agent-Alice and Agent-Bob's. For Agent-Bob's loss (see Equation. 8.3), the Mean Square Error (MSE) distance between **m** and **m'** is used. It is then written as:

$$\mathcal{L}_{Bob} = \left(\sum_{i=1}^{m}(m_i - m'_i)^2\right)/m\,, \tag{8.5}$$

which is equivalent to:

$$\mathcal{L}_{Bob} = ((\mathbf{s}^{(\mathbf{m})} - \mathbf{s}^{(\mathbf{m}')}) \odot \Omega)^2/m\,, \tag{8.6}$$

with $\odot$ the point-wise operation.

Agent-Alice uses the loss presented in Equation. 8.4 with $dist(\mathbf{x},\mathbf{y})$ calculated as:

$$dist(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{w}\sum_{j=1}^{h}(x_{ij} - y_{ij})^2\right)/w.h\,, \tag{8.7}$$

and thus Agent-Alice's loss is:

$$\mathcal{L}_{Alice} = \lambda_A \cdot dist(\mathbf{x},\mathbf{y}) + \lambda_B \cdot \mathcal{L}_{Bob} - \lambda_E \cdot \mathcal{L}_{Eve} \tag{8.8}$$

where $\lambda_A$, $\lambda_B$, $\lambda_E \in$ belongs to [0,1] and sum to one in order to adjust the contribution of each term to the loss. In this architecture, the magnitude of modifications is not restricted and therefore, the stego noise power is too strong. To impose a stego noise power restriction, we propose a second architecture.

### 8.4.4 Second-Architecture (noise power reduction)

The second architecture improves the first one by imposing a stronger restriction on the magnitude of modification of the stego noise. We force Agent-Alice to

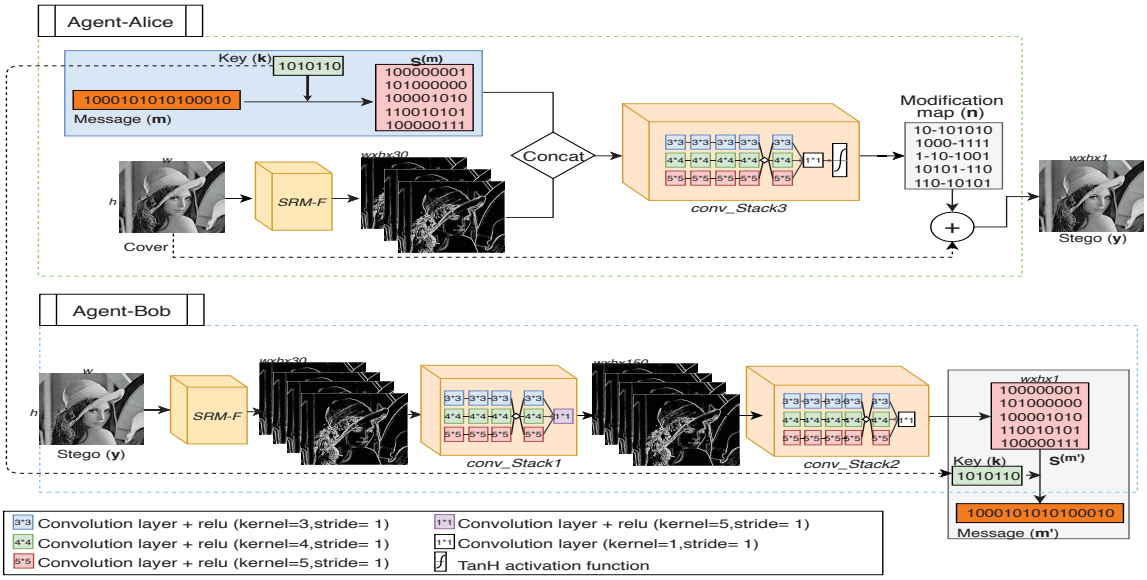



FIGURE 8.4: Agent-Alice and Agent-Bob with the second architecture.

make the least changes to the cover while Agent-Bob is still able to retrieve the secret message correctly. Within this architecture, see Figure. 8.4, Agent-Bob's design remains the same as in the previous architecture. However, the architecture of Agent-Alice is changed. Instead of letting the network decide the intensity of modification for each pixel in the cover image, it is restricted to a ternary modification; thus the stego noise values are {-1, 0, 1}. More precisely, during the first iterations, values are in the range $[-1, 1]$ and belong to $\mathbb{R}$. But at the end of iterations, a discretization is done in order to have only three discrete values {-1, 0, 1}.

Said differently, Agent-Alice generates a modification map $\mathbf{n} \in \{-1, 0, 1\}$ which is then added to the cover image $\mathbf{x}$ to generate the stego $\mathbf{y}$ directly (see Figure. 8.4). The generation of $\mathbf{n}$ is performed using the resulting feature maps of *SRM-F*, and the spread message $\mathbf{s}^{(\mathbf{m})}$. These are concatenated and fed to *conv_Stack3*; the output of the latter is $\mathbf{n}$ whose values, thanks to a TanH activation function, are in the range $[-1, 1]$. Once the discretization is activated, $\mathbf{n}$ values are one of the three discrete values {-1, 0, 1}.

Agent-Bob's network loss remains the same as in Equation. 8.5. For Agent-Alice's loss, we calculate the mean of the absolute values of the modification maps $\mathbf{n}$ (which is equivalent to the MSE) for the distance between the cover image $\mathbf{x}$

and the stego $\mathbf{y}$.

$$dist(\mathbf{x}, \mathbf{y}) = \left( \sum_{i=1}^{w} \sum_{j=1}^{h} (|n_{ij}|) \right) / w.h \,, \tag{8.9}$$

One can notice that minimizing the loss using this distance forces Agent-Alice to output only **zeros** over the map of modifications. Agent-Bob is then no longer capable of retrieving the secret message. To reduce this constraint, we introduce a constant $\beta$ in Agent-Alice's loss. This $\beta$ value is related to the change rate notion. The loss thus becomes:

$$\mathcal{L}_{Alice} = \lambda_A \cdot (dist(\mathbf{x},\mathbf{y}) - \beta) + \lambda_B \cdot \mathcal{L}_{Bob} - \lambda_E \cdot \mathcal{L}_{Eve}, \tag{8.10}$$

where $\lambda_A, \lambda_B, \lambda_E \in [0, 1]$. Note that $\beta$ controls the discretion of the embedding network, i.e how many pixels Agent-Alice is allowed to alter from the cover image $\mathbf{x}$.

### 8.4.5 Third-Architecture (source separation)

The architecture presented in Figure. 8.5 is proposed as an improvement to the second architecture. The embedding part of the second architecture was changed to make as few changes as possible. The extracting part, on the other hand, was kept as it is from the first architecture. Nevertheless, constraining the amplitude of modifications directly impacts the extracting part. When we limit the number of pixels that can be modified, more errors occur during message extraction. In other words, altering fewer pixels means less detectability, but more errors during *message-extraction*, while changing more pixels means fewer errors when retrieving the message, but more detectability. How can Agent-Bob extract the secret message correctly when Agent-Alice carries out the minimal required modification?

In embedding algorithms such as S-UNIWARD [Holub et al., 2014], WOW [Holub and Fridrich, 2012], etc, the message coding requires, in practice, the use of a Syndrome-Trellis-Codes STC (described in ection. 3.3.2.2. The extractor (Bob)

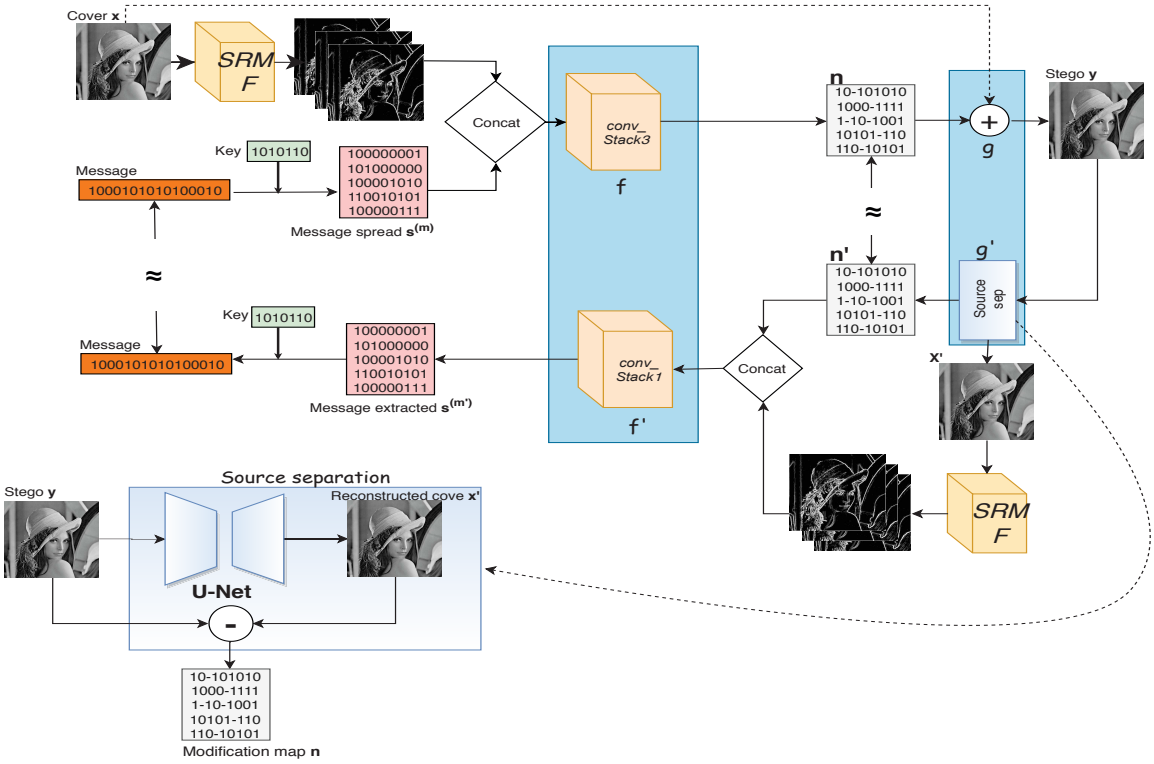



FIGURE 8.5: Agent-Alice and Agent-Bob with the third architecture.

has access to the parity-check matrix $\mathbf{h} \in \{0, 1\}^{w \times h}$ used by Alice during the embedding process. This matrix $\mathbf{h}$ is then shared between Alice and Bob, so Bob can easily retrieve the message from $\mathbf{y}$ by calculating the matrix product as:

$$\mathbf{m} = \mathbf{h} \cdot \text{lsb}(\mathbf{y}),$$

with lsb (.) the function extracting the LSB plane.

In our proposed method, no parity-check matrix is shared between Agent-Alice and Agent-Bob, nor any related information. If Agent-Bob has to mimic an STC to perform the message extraction in the *3-player game*, it should learn a matrix in conjunction with Agent-Alice in order to retrieve the message correctly. Without any topological or mathematical construction, this can be difficult, especially when the system used has many loss terms to minimize (Nash equilibrium issue).

A solution for that problem could be to inject a coding/decoding block inside Agent-Alice and Agent-Bob. Nevertheless, this is not an easy task, and before

trying this solution, we preferred exploring the impact of increasing the link between the two agents. The integration of a coding and decoding block is postponed for future work.

We continue in the path of increasing the link between the Agent-Alice and Agent-Bob by forcing their topology to be more anti-symmetric. Referring to Figure. 8.5, we draw two blue rectangles to show where this anti-symmetry has been injected. On one of the rectangle, we noted on the figure, $g$, the *point-wise summation* block, and $g$*'*, the *sources separation* block. $g$ and $g$*'* can be seen as inverse functions such that:

$$\begin{aligned} g : \{-1, 0, +1\}^{w \times h} \times \mathbb{R}^{w \times h} &\rightarrow \mathbb{R}^{w \times h} \\ g(\mathbf{n}, \mathbf{x}) &= \mathbf{n} + \mathbf{x} = \mathbf{y} \\ g' : \mathbb{R}^{w \times h} &\rightarrow \{-1, 0, +1\}^{w \times h} \times \mathbb{R}^{w \times h} \\ g'(\mathbf{y}) &= (\mathbf{n}', \mathbf{x}') \end{aligned}$$

For a given cover, $\mathbf{x}$, and a stego noise, $\mathbf{n}$, $g'((g(\mathbf{n}, \mathbf{x}))$ gives an estimated cover, $\mathbf{x}$**'**, and a estimated stego noise, $\mathbf{n}$**'**. During the learning, we then minimize the distance between $\mathbf{x}$ and $\mathbf{x}$**'** (which is equivalent to minimize the distance between $\mathbf{n}$ and $\mathbf{n}$**'**); See Equation. 8.11.

The *sources separation* block act as a denoiser. The well-known U-Net [Ronneberger et al., 2015] has thus been used and integrated into Agent-Bob's architecture (see Figure. 8.5). The goal of the U-Net is to reconstruct the cover image $\mathbf{x}$ from the stego $\mathbf{y}$ image. The modification map $\mathbf{n}$ is resulting from the subtraction of the reconstructed cover $\mathbf{x}$**'** from the stego image $\mathbf{y}$.

Referring to Figure. 8.5, on the other rectangle, we noted on the figure, $f$, the *conv_Stack3* block, and $f$*'*, the *conv_Stack1* blocks. $f$ and $f$*'* can be seen as inverse functions from the point view of the spread message $\mathbf{s}^{(m)}$ and the modification map

$\mathbf{n}$ :

$$f : \mathbb{R}^{w \times h \times 30} \times \{0,1\}^{w \times h} \rightarrow \{-1,0,+1\}^{w \times h}$$
$$f_{\mathbf{r}}(\mathbf{s}^{(\mathbf{m})}) = \mathbf{n}$$
$$f' : \mathbb{R}^{w \times h \times 30} \times \{-1,0,+1\}^{w \times h} \rightarrow \{0,1\}^{w \times h}$$
$$f'_{\mathbf{r}'}(\mathbf{n}') = \mathbf{s}'^{(m')}$$

with $\mathbf{r}$ and $\mathbf{r'}$ the 30 residual images obtained by *SRM-F* filtering.

For a given spread message $\mathbf{s}^{(m)}$, and residual images $\mathbf{r}$ and $\mathbf{r'}$, $f'_{\mathbf{r}'}((f_{\mathbf{r}}(\mathbf{s}^{(\mathbf{m})}))$ gives an estimated spread message $\mathbf{s}'^{(m')}$. During the learning, we then minimize the distance between $\mathbf{m}$ and $\mathbf{m'}$ (which is equivalent to minimize the distance between $\mathbf{s}^{(m)}$ and $\mathbf{s}'^{(m')}$; see Equation. 8.6); See Equation. 8.12. Note that *conv_Stack1* and *conv_Stack3* are the same than in the second architecture.

Agent-Bob's loss is thus composed of two loss terms:

- the cover reconstruction loss:

$$\mathcal{L}_{cover\_recons} = MSE(\mathbf{x'}, \mathbf{x}) \tag{8.11}$$

- the message extraction loss:

$$\mathcal{L}_{message\_extract} = MSE(\mathbf{m}, \mathbf{m'}) \tag{8.12}$$

The Agent-Bob loss is thus given by:

$$\mathcal{L}_{Bob} = 1/2(\mathcal{L}_{cover\_recons} + \mathcal{L}_{message\_extract}) \tag{8.13}$$

## 8.5 Experiments

### 8.5.1 Dataset and software platform

The experiments are carried out on two image sources. The first is the BOSSBase database 1.01 [Bas et al., 2011]. Images of this database offer different texture characteristics, which explains why it is widely used in steganalysis. BOWS2 [Bas and Furon, 2008] is our second database.

Due to our GPU computing platform, time limitation and the *3-player game* nature, which takes a lot of time to train due to the two-phase learning, we conduct all the experiments on images of 256×256 pixels. To this end, we resampled all images from the two databases from 512×512 pixels to 256×256 pixels, using the imresize() Matlab function with default parameters.
We implemented the proposed architectures presented in Section. 8.4 using TensorFlow V 1.6. As for comparison, we use S-UNIWARD [Holub et al., 2014], and WOW [Holub and Fridrich, 2012], two content-adaptive methods for spatial domain embedding. All our experiments were conducted on NVIDIA Titan X GPU platform.

### 8.5.2 Training, Validation, Test

We start the training phase by preparing a pre-trained model of Agent-Eve. For this, we use the S-Uniward algorithm to generate 10,000 stegos from the BOSSBase. 10,000 cover/stego pairs are then obtained. We use 4,000 pairs to train a Yedroudj-Net network. After several epochs of training, we obtain a learned model of Yedroudj-Net, which we transfer to Agent-Eve; thus, Agent-Eve does not learn from scratch.

Once Agent-Eve is pre-trained, we start the learning phase of the 3-players (Agent-Alice, Agent-Bob and Agent-Eve). We use BOSSBase as the image source. The

messages are generated using a PRNG [2] with a chosen payload. We use Adam, an adaptive optimizer, to train our system of three networks where the mini-batch size is set to 4. The values of $\lambda_A$, $\lambda_B$, $\lambda_E$ are set to 0.2, 0.4, and 0,4 respectively. These values are chosen empirically; they thus may not be optimum.

During training, we set the maximum number of iterations *max-iter* to 1 million iterations. The training is alternated between Agent-Alice and Agent-Bob from one side, and Agent-Eve from another side. We set *it1* to 50 iterations, and *it2* to 1 iteration (see Algorithm. 1, so Agent-Alice and Agent-Bob are trained for 50 iterations, compared to 1 iteration for Agent-Eve).

When the losses of the three networks tend to be stable, we freeze their training, and we integrate the discretization module into Agent-Alice (see Figure. 4.6). Then, we resume the training for several iterations (more or less 50,000 iterations), so the system learns how to generate stego images with discrete pixel values.

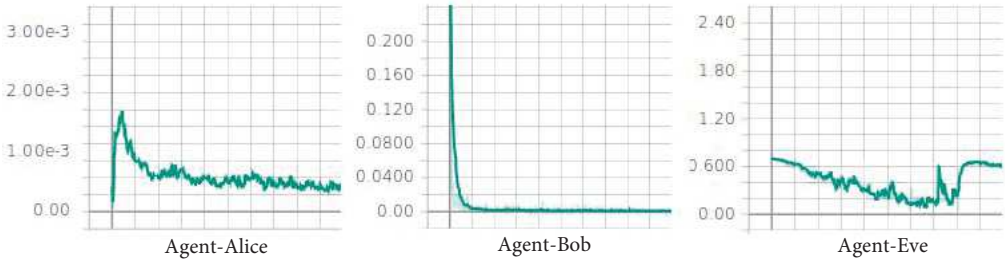



FIGURE 8.6: Generic loss evolution of Agent-Alice, Agent-Bob and Agent-Eve after 300000 iterations (first architecture).

We should note that activating the discretization module at the beginning of the training phase prevents the system from converging (as the round function is not differentiable). We, therefore, let the system converge towards a solution, then we force it to work on images with discrete pixel values. Once our system is well trained (losses curves are stable, as shown in Figure. 8.6), we stop the training phase.

To evaluate the performance of our method, we measure the *security* and the *transmission error*. The security is measured with the probability of error (Pe) of

[2]PRNG: Pseudo-Random Number Generator. The PRNG generates a binary vector **m** whose values 0 and 1 are uniformly distributed.

a given steganalyzer, where Pe is computed as the average of the false alarm rate and the missed detection rate. For the transmission error, we use the Bit Error Rate (BER). This represents the number of erroneous bits received by Bob. Note that the BER should not be used in a standard steganographic scenario since no channel error should be considered during the transmission of the stego image. Unfortunately, none of the architecture of existing literature for the *3-player game* are able to embed a message that could be retrieved without errors. In practical use, one should consider a preliminary coding of the message by a correcting code in order to ensure a correct extraction from Bob's side. We will not integrate such a coding but will discuss it. Indeed, this work is primarily on the 3-player concept definition and the proposition of three architectures.

Therefore for the testing phase, Agent-Alice starts generating 10,000 Stego images from BOWS2, where a random message is embedded in a cover image using a random key. The generated images are used to evaluate the accuracy of Bob's message retrieval on the one hand, and the security of our steganographic system against the steganalyzer Yedroudj-Net on the other. Please note that the steganalysis with Yedroudj-Net (learning and testing) is done on BOWS2Base, as for the training of the 3-players (Agent-Alice, Agent-Bob, and Agent-Eve) it is carried out using the different and disjointed BOSSBase. The results are presented in the following subsection.

### 8.5.3 Results of the three architectures

**First architecture:**

In Table. 8.1, we report the Bit Error Rate (BER) and the Probability of error (Pe) obtained using the first architecture. These tests are carried on BOWS2 database at different payloads 0.2 bpp, 0.4 bpp and 1 bpp. The steganalyzer used is Yedroudj-Net. Regardless of the payload, Bob, who uses the Agent-Bob, can correctly recover the secret message with a BER equal to zero. We can observe

| Payload | Bit Error Rate | Probality of error (Pe) |
|---|---|---|
| 0.2 | 0 | 3.7% |
| 0.4 | 0 | 3.3% |
| 1 | 0 | 2.5% |

TABLE 8.1: First architecture BER and Pe for different payloads.

that this architecture is not at all secure since the detection accuracy obtained using Yedroudj-Net is between 96% and almost 98%.

The first architecture offers a good embedding capacity, where we can perfectly recover the secret message even when the payload is significant, although the low security given by this architecture does not make it interesting for steganography purposes.

Note that this architecture is the closest form of architecture to GSIVAT or HiDDeN since the main principle is to spread the message in the spirit of a spread spectrum watermarking approach. This first architecture, GSIVAT, and HiDDeN are definitively unsecure approaches since the stego noise power is too strong. We investigated in the second architecture a way to constrain pixels modifications to -1 or +1.

**Second architecture:**

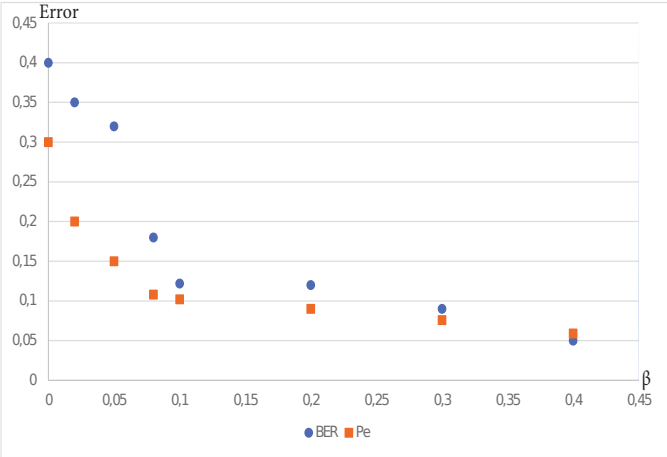



FIGURE 8.7: BER and Pe for the second architecture for a payload size of 0,4 bpp in function of change rate $\beta$.

In the second architecture, we manage to control the noise power introduced by Agent-Alice during the embedding process. For this purpose, we have conducted several experiments with different values of the change rate $\beta$ of the Equation. 8.10. In Figure. 8.7, we present the Bit Error Rate (BER) and the Probability of error (Pe) of the second architecture, when different $\beta$ values are used, for a fixed payload of 0.4 bpp. The used steganalyzer is Yedroudj-Net. We can see that, for $\beta$ equal to 0.4, Eve, the steganalyzer (Yedroudj-net) gets a probability of error (Pe) close to 6% when Bob makes 5% of Bit Error Rate (BER). When $\beta$ is set to 0.2, the security of our system improves with a Pe equal to 9%, although the message extraction becomes more difficult for Bob with 12% of Bit Error Rate.

For a $\beta$ value equal to or less than 0.05, the generated images are more secure. Nevertheless, the BER increases considerably. The BER is 32% and 40%, and the Pe is 15% and 30% for $\beta$ equals 0,05 and 0 respectively. Those BER values are very high and would be difficult to correct with a correcting code without strongly reducing to the real payload size. The best $\beta$ value is those that in the same time, maximize the detection error value (Pe), and minimize the size of the coded message; the message has thus to be encoded such that a correct extraction is ensured. Since those considerations are a little bit out of topic, we propose a rapid analysis by choosing a particular point where there is a sudden variation of BER (i.e. the inflection point). For this architecture and for a payload size 0.4 bpp, the value of $\beta$ corresponding to a sudden variation of BER is around $\beta = 0.1$. At $\beta = 0.1$, the BER=12%. Using the [7,4,3] Hamming Error Correcting Code (ECC) ensures to correct at most a BER of 14%, so we would, on average, correct all the errors. With this ECC the *real* payload size is thus 0.23 bpp, and we measure for that point a probability of error of 10.2%. It is clear that this architecture gives better results compared to the first one, where for a payload of 0.2 bpp the Pe is 3%. Nevertheless, those results are still not convincing, and as explained before, the problem of the second architecture is the weak relationship between Agent-Alice and Agent-Bob. The third architecture aims to counter this weakness.

FIGURE 8.8: (Left) The BOSSBase cover image. (Middle) the corresponding stego images with 0.4 bpp and $\beta = 0.1$ using the second architecture. (right) the modification maps between the cover image and the corresponding stego where black=0, white=+/- 1.

In Figure. 8.8 we can observe that this architecture can learn to concentrate the embedding on textured regions, which are more difficult for a steganalyzer to detect. This architecture has the capacity to learn and find interesting zones to implement steganography.

**Third architecture:**

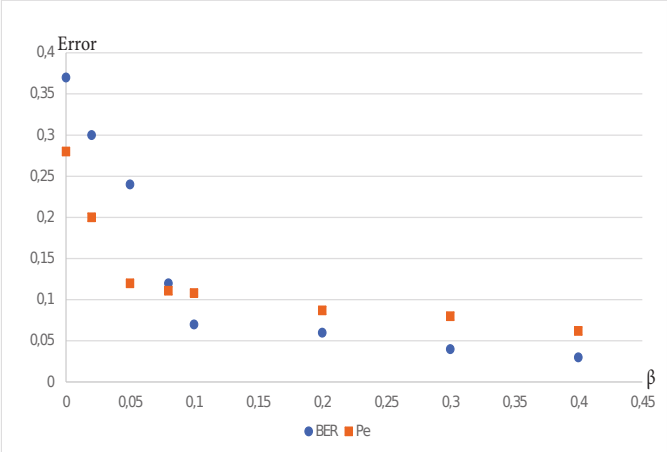



FIGURE 8.9: BER and Pe for the third architecture for a payload size of 0.4 bpp in function of change rate $\beta$.

As mentioned above, the third architecture has been proposed to improve the extraction part of the second architecture. However, this requires more time to converge, but the bright side is that it offers a better message retrieval accuracy. In Figure. 8.9, we present the bit error rate (BER) and the probability of error

(Pe) of the third architecture as a function of $\beta$ the change rate, for a fixed payload 0.4 bpp.

Compared to the second architecture, the error of Bob (BER) is smaller regardless of the value of $\beta$, even though the detection accuracy remains almost the same for both architectures. The BER obtained with the third architecture and compared to the second architecture is 2% lower when $\beta = 0.4$, 6% lower when we set $\beta = 0.2$, and 5% lower when $\beta = 0.1$. By observing these results, we see that Agent-Alice and Agent-Bob have nevertheless benefited from the third architecture. More joint learning between Agent-Alice and Agent-Bob seems to be the right research direction.

Looking at Figure. 8.9, similarly to the second architecture, the inflection point is around $\beta = 0.1$. At this point, the BER value is 0.06. The errors can be corrected with a Hamming code [15,11,3]. In this case the *real* payload size is $(0, 4 \cdot 11)/15 = 0.293bpp \approx 0.3$ bpp. Using this payload we get a Pe of about 11%.

To give a comparison, we run a steganalysis with Yedroudj-Net against two steganographic algorithms S-UNIWARD and WOW at a payload size of 0.3 bpp using the database BOWS2. The probability of error (Pe) is 27.3% (resp. 22.4%) for S-UNIWARD (resp. WOW). There is undoubtedly a security gap with the actual embedding schemes, but again, the objective is to define the *3-player game* concept, and to analysis, its potential compare to modern embedding scheme and steganalysis scheme. In that, the third architecture shows that there is a real potential, and these experiments pave the way to many research.

Figure. 8.10 shows the modifications zone using the third architecture. We can observe some adaptivity, as the embedding is more dense in textured zones.

The results show that this architecture can implement adaptive embedding. It also indicates that the link between Agent-Alice and Agent-Bob should be reinforced in the future to reduce the BER and eventually reduce the change rate and increase the security level. These results should not deter from the 3-player concepts, even if the security level at the moment is lower compared to the traditional adaptive

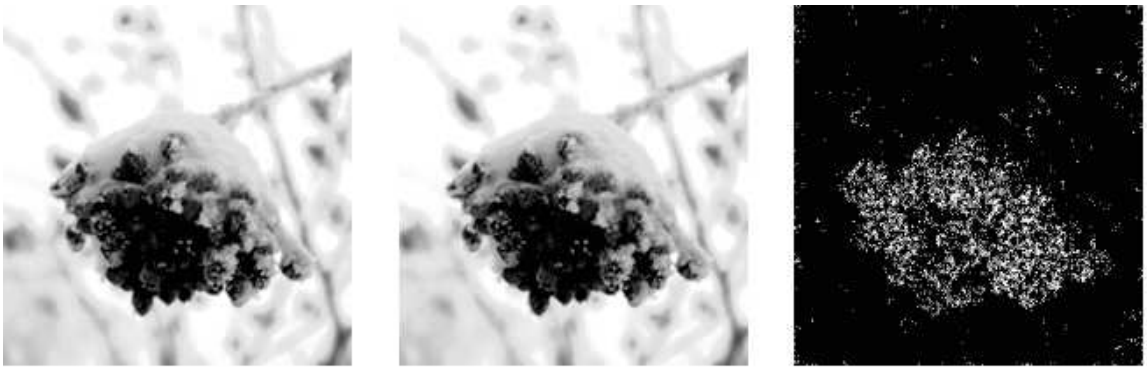


FIGURE 8.10: (Left) The BOSSBase cover image. (Middle) The stego images with payload size 0.4 bpp and $\beta = 0.1$ using the third architecture. (right) the modification maps between the cover image and the corresponding stego where black=0, white=+/- 1.

embedding algorithms such as S-UNIWARD, WOW. Indeed, our proposed method shows real improvements compared to GSIVAT and HiDDEeN.

## 8.6 Conclusion and perspectives

In this chapter, we have first recalled the *3-player game approach* and defined the general concept and how to correctly use it for steganography. Three architectures based on the *3-player game approach* have been proposed. The first architecture came to fix the flaws made in GSIVAT and HiDDEeN. Nevertheless, the first architectures behave similarly to GSIVAT and HiDDEeN. It is not adapted for steganography purposes due to its extreme detectability. With the second architecture, we suggested a new way to embed a message in the cover image. Instead of directly modifying the cover image, which implies a significant noise power signal, we proposed to generate a modification map with values belonging to {-1,0,1}. The stego is then generated by adding the modification map to the cover. This architecture is much more secure than the first one, but it generates more errors during message extraction. Finally, the third architecture imposes more joint learning between Agent-Alice and Agent-Bob in order to reduce the errors during the message extraction. This third architecture takes more time to converge but achieves better results.

The third architecture, with a *real* payload size of 0,3 bpp, can perfectly retrieve an embedded message (with the help of error-correcting code), for a security level Pe = 10,8%. Even if the obtained results do not surpass current state-of-the-art embedding algorithms, these results show the extent of the potential of such method. The major contribution is thus to propose a formalization of the *3-player game* concept, and an end-to-end method using neural networks that can learn to simulate algorithms using human-based rules (steganography and steganalysis).

We expect this work to lead to fruitful avenues for further research. In future work, we can study the possibility of synchronizing more Agent-Alice and Agent-Bob, and thus improving the general performances. We can also try using a more subtle loss function. This can help the networks to converge to a better solution. Furthermore, finding a theoretical way to compute the change rate $\beta$ can help to accelerate the learning process. The values of $\lambda_A$, $\lambda_B$, $\lambda_E$ could also be more extensively studied.

# Chapter 9

# Conclusions and Perspectives

## Contents

## 9.1 Conclusions

In the last few years, both steganalysis and steganography have witnessed a rapid evolution as a result of the progress made in coding/learning of machine learning methods. This thesis contributes to this quick field development.

### 9.1.1 Steganalysis

With regard to steganalysis, we have presented a new method for the spatial domain steganalysis. This method is a deep learning-based approach, more specifically a CNN-based steganalysis. The architecture of this proposed CNN is considered as a shallow one. It is composed of a *pre-processing module*, five *convolutional blocks* and a classification module made of three fully connected layers. This network is referred to as "Yedroudj-Net" CNN. In terms of performance, Yedroudj-Net achieves superior results to those of the other steganalyzers existing in mid-2018, thereby making it state-of-the-art of this period. In an effort to achieve even better results, we used Yedroudj-Net with two well-known complements that generally improve the performances of the deep learning model. The first complement is data augmentation. To that end, different ways of data augmentation have been tested in order to find the best practical way to do it in the context of steganalysis. Data augmentation made it possible to reduce the Yedroudj-Net error significantly. The second tested add-on is the use of a set of CNNs. If the use of a single network yields good results, the use of a set of networks gives even better results (bootstrap or aggregation). This has been experimentally proven where we were able to obtain 3 to 6% reduction of the detection error with an ensemble made of three Yedroudj-Nets.

Steganalysis using CNN's is a novel framework that offers several promising directions for future research. For short-term research, updating Yedroudj-Net to Yedroudj-Net-V2.0 may be a possible path to explore. Among the many improvements that can be introduced to our network, we mention the reduction of the size of the classification module while increasing the number of the convolutional

layer. In this context, it may also be beneficial to change the simple convolutions layers by depth-wise and residual convolutional layer. This helps to minimize the complexity of the Yedroudj-Net network and thus makes it even faster to train. Another improvement is to make Yedroudj-Net-V2.0 works on both spatial and the frequency domain (JPEG steganalysis). Last but not least, we could evaluate the transfer learning technique. This last can be used to handle stegos with a small payload. As for long-term research, one possible direction is what we can names as *limitless learning set* scenario. This scenario consists of using the power of GAN models to generate new examples "that look like" "real" examples, more precisely, examples that have a close distribution to the one of the learning set images. With such a solution, the "real world situation" is no longer a problem since, even with a low learning regime, we would be able to generate as many images as required to obtain satisfactory results by the steganalyst model. This path raises another important question. If the scenario *limitless learning set* does not require any learning set to begin with (as it is automatically generated from the test set), thus, both learning and test sets have a similar distribution. So can this scenario provide the ultimate solution to the cover source mismatch?

### 9.1.2 Steganography

As for steganography, and with the objective of shifting from adaptive steganography to strategic adaptive steganography, we have defined a new way for embedding which is based on the use of a generative adversarial network (GAN) and the 3-player concept. Our contribution has therefore been to develop a strategic adaptive approach for embedding secret data, which makes it possible to take into account not only the distribution of the current cover image but also the evolution of the steganalyzer (attacker).

Our proposition is mainly based on the use of three networks (embedding network, extracting network and steganalysis network) that competes against each other to learn an embedding model. For that, we have proposed three different architectures, all based on the *3-player-game concept*. The first architecture was proposed

as a rigorous alternative to other existing methods. However, this architecture is not adapted for steganography purposes due to its extreme detectability. As for the other architecture, we suggested a new way to embed a message in the cover image. Instead of directly modifying the cover image, which implies a significant noise power signal, we proposed to generate a modification map with values belonging to {-1,0,1}. The stego is then generated by adding the modification map to the cover. This architecture is much more secure than the first one, but it produces more errors at message extraction. The third architecture enriches the second one with better interaction between the embedding and extracting networks.

Although the obtained results are no better than the most recent embedding algorithms, our proposal demonstrates the potential for this method. The main contributions are therefore to offer a formalization of the 3-player game concept and an end-to-end approach based on the use of neural networks, which make it possible to learn how to simulate algorithms based on human rules (steganography and steganalysis).

Following this brief synthesis of the work presented in this manuscript, it seems reasonable to question the different possibilities for improving this work. To this end, several potential paths could be explored.

As pointed in section. 8.6, approaches of the 3-player game suffer from the lack of synchronization between Agent-Alice and Agent-Bob. More specifically, Agent-Bob, whose task is to extract the secret message, has access to the same information as Agent-Eve (except the key), although the Agent-Eve's task is much easier. This explains the difficulties encountered by Agent-Bob when extracting the embedded message.

To overcome this problem, two possible solutions are to be investigated. The first is to synchronize more Agent-Alice and Agent-Bob by using more constraints on the training of their networks. The second is to inject a coding/decoding block inside Agent-Alice and Agent-Bob. In order for them to simulate the process of an STC.

Another possible direction for future research is to explore the possibility of combining two (or more) steganography with GAN approaches to forge a new one. Take, for example, the approach with probability map generation. This approach can benefit from the use of adversarial embedding iterated ideas when it comes to generating the change probability map.

# Bibliography

M. Abadi and D. G. Andersen. Learning to Protect Communications with Adversarial Neural Cryptography. In *ArXiv; Rejected from the 5th International Conference on Learning Representations, ICLR'2017.*, volume abs/1610.06918, 2016. URL http://arxiv.org/abs/1610.06918.

R. J. Anderson and F. A. P. Petitcolas. On the limits of steganography. *IEEE Journal on Selected Areas in Communications*, 16(4):474–481, 1998.

P. Bas and T. Furon. BOWS-2 Contest (Break Our Watermarking System), 2008. Organized between the 17th of July 2007 and the 17th of April 2008. http://bows2.ec-lille.fr/.

P. Bas, T. Filler, and T. Pevný. 'Break Our Steganographic System': The Ins and Outs of Organizing BOSS. In *Proceedings of the 13th International Conference on Information Hiding, IH'2011*, volume 6958 of *Lecture Notes in Computer Science*, pages 59–70, Prague, Czech Republic, May 2011. Springer.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

D. Borghys, P. Bas, and H. Bruyninckx. Facing the cover-source mismatch on jphide using training-set design. In *Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '18, pages 17–22, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5625-1.

M. Boroumand and J. Fridrich. Nonlinear feature normalization in steganalysis. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia*

*Security*, IH&MMSec '17, pages 45–54, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5061-7.

M. Boroumand, M. Chen, and J. J. Fridrich. Deep residual network for steganalysis of digital images. *IEEE Trans. Information Forensics and Security*, 14(5):1181–1193, 2019.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

C. Cachin. An information-theoretic model for steganography. In *Information Hiding, Second International Workshop, Portland, Oregon, USA, April 14-17, 1998, Proceedings*, pages 306–318, 1998.

G. Cancelli, G. J. Doërr, M. Barni, and I. J. Cox. A comparative study of ±1 steganalyzers. *2008 IEEE 10th Workshop on Multimedia Signal Processing*, pages 791–796, 2008.

M. Chaumont. Deep Learning in steganography and steganalysis from 2015 to 2018. In M. Hassaballah, editor, *Digital Media Steganography: Principles, Algorithms, Advances*, volume abs/1904.01444, page 39. Elsevier, 2020. URL http://arxiv.org/abs/1904.01444. This chapter will appear in 2020.

M. Chen, V. Sedighi, M. Boroumand, and J. Fridrich. JPEG-Phase-Aware Convolutional Neural Network for Steganalysis of JPEG Images. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec'17, page 10, Drexel University in Philadelphia, PA, June 2017.

M. Chen, M. Boroumand, and J. J. Fridrich. Reference channels for steganalysis of images with convolutional neural networks. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec 2015, Paris, France, July 3 - 5, 2019*, 2019.

R. Cogranne, V. Sedighi, J. Fridrich, and T. Pevný. Is ensemble classifier needed for steganalysis in high-dimensional feature spaces? In *2015 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, Nov 2015. doi: 10.1109/WIFS.2015.7368597.

I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich, and T. Kalker. *Digital Watermarking and Steganography (Second Edition)*. Morgan kaufmann, 2007.

R. Crandall. Some notes on steganography. *Posted on steganography mailing list*, pages 1–6, 1998.

S. Craver. On public-key steganography in the presence of an active warden. In *Information Hiding*, 1998.

T. Denemark, V. Sedighi, V. Holub, R. Cogranne, and J. Fridrich. Selection-channel-aware rich model for steganalysis of digital images. In *Proceedings of IEEE International Workshop on Information Forensics and Security, WIFS'2014*, pages 48–53, Atlanta, Georgia, USA, Dec. 2014.

T. Denemark, M. Boroumand, and J. Fridrich. Steganalysis features for content-adaptive jpeg steganography. *IEEE Transactions on Information Forensics and Security*, 11(8):1736–1746, Aug. 2016a.

T. Denemark, J. J. Fridrich, and P. C. Alfaro. Improving selection-channel-aware steganalysis features. In *Media Watermarking, Security, and Forensics 2016, San Francisco, California, USA, February 14-18, 2016*, pages 1–8, 2016b.

T. Denemark, P. Bas, and J. J. Fridrich. Natural steganography in jpeg compressed images. In *Media Watermarking, Security, and Forensics*, 2018.

X. Deng, B. Chen, W. Luo, and D. Luo. Fast and effective global covariance pooling network for image steganalysis. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec'19, pages 230–234, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6821-6.

J. C. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

M. Ettinger. Steganalysis and game equilibria. In *Information Hiding, Second International Workshop, Portland, Oregon, USA, April 14-17, 1998, Proceedings*, pages 319–328, 1998.

K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 1625–1634, 2018.

T. Filler and J. J. Fridrich. Gibbs construction in steganography. *IEEE Trans. Information Forensics and Security*, 5(4):705–720, 2010.

T. Filler, J. Judas, and J. J. Fridrich. Minimizing embedding impact in steganography using trellis-coded quantization. In *Media Forensics and Security II, part of the IS&T-SPIE Electronic Imaging Symposium, San Jose, CA, USA, January 18-20, 2010, Proceedings*, page 754105, 2010.

T. Filler, J. Judas, and J. J. Fridrich. Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Trans. Information Forensics and Security*, 6(3-2):920–935, 2011.

J. Fridrich. *Steganography in Digital Media*. Cambridge University Press, 2009. ISBN 9781139192903. Cambridge Books Online.

J. Fridrich and J. Kodovský. Rich Models for Steganalysis of Digital Images. *IEEE Transactions on Information Forensics and Security, TIFS*, 7(3):868–882, June 2012.

J. Fridrich and J. Kodovský. Steganalysis of lsb replacement using parity-aware features. In *Proceedings of the 14th International Conference on Information Hiding*, IH'12, pages 31–45, 2013. ISBN 978-3-642-36372-6.

J. Fridrich, M. Goljan, and D. Hogea. Attacking the outguess, 2002.

J. J. Fridrich. Feature-based steganalysis for JPEG images and its implications for future design of steganographic schemes. In *Information Hiding, 6th International Workshop, IH 2004, Toronto, Canada, May 23-25, 2004, Revised Selected Papers*, pages 67–81, 2004.

J. J. Fridrich and T. Filler. Practical methods for minimizing embedding impact in steganography. In *Security, Steganography, and Watermarking of Multimedia Contents IX, San Jose, CA, USA, January 28, 2007*, page 650502, 2007.

J. J. Fridrich, M. Goljan, P. Lisonek, and D. Soukal. Writing on wet paper. In *Security, Steganography, and Watermarking of Multimedia Contents VII, San Jose, California, USA, January 17-20, 2005, Proceedings*, pages 328–340, 2005.

J. J. Fridrich, J. Kodovský, V. Holub, and M. Goljan. Breaking HUGO - the process discovery. In *Information Hiding - 13th International Conference, IH 2011, Prague, Czech Republic, May 18-20, 2011, Revised Selected Papers*, pages 85–101, 2011.

K. Fukushima. "cognitron: A self-organizing multilayered neural network". *"Biological Cybernetics"*, 20(3):121–136, Sep 1975. ISSN 1432-0770. doi: 10.1007/BF00342633. URL https://doi.org/10.1007/BF00342633.

S. Ghannay. *Etude sur les representations continues de mots appliquees a la detection automatique des erreurs de reconnaissance de la parole*. PhD thesis, 2017. URL http://www.theses.fr/2017LEMA1019. Thèse de doctorat dirigée par Estève, Yannick et Camelin, Nathalie Informatique Le Mans 2017.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS'2010*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

I. J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *10th IEEE International Conference on Computer Vision (ICCV 2005), 17-20 October 2005, Beijing, China*, pages 1458–1465, 2005.

P. Guillot. *La cryptologie: l'art des codes secret: L'art des codes secrets*. EDP sciences, 2013.

J. Hayes and G. Danezis. Generating Steganographic Images Via Adversarial Training. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings ofAdvances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NIPS'2017*, pages 1951–1960, Dec. 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37 (9):1904–1916, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR'2016*, pages 770–778, Las Vegas, Nevada, June 2016.

M. A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998. ISSN 1541-1672.

G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

V. Holub and J. Fridrich. Designing Steganographic Distortion Using Directional Filters. In *Proceedings of the IEEE International Workshop on Information Forensics and Security, WIFS'2012*, pages 234–239, Tenerife, Spain, Dec. 2012.

V. Holub, J. J. Fridrich, and T. Denemark. Random projections of residuals as an alternative to co-occurrences in steganalysis. In *Media Watermarking, Security, and Forensics 2013, Burlingame, CA, USA, February 5-7, 2013, Proceedings*, page 86650L, 2013.

V. Holub, J. Fridrich, and T. Denemark. Universal Distortion Function for Steganography in an Arbitrary Domain. *EURASIP Journal on Information Security, JIS*, 2014(1):1, Jan. 2014.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79 (8):2554–2558, 1982. ISSN 0027-8424. doi: 10.1073/pnas.79.8.2554.

J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81(10):3088–3092, 1984. doi: 10.1073/pnas.81.10.3088.

D. Hu, L. Wang, W. Jiang, S. Zheng, and B. Li. A novel image steganography method via deep convolutional generative adversarial networks. *IEEE Access*, 6:38303–38314, 2018.

G. A. F. III and T. S. Gomez. Steganography obliterator: an attack on the least significant bits. In *Proceedings of the 3rd Annual Conference on Information Security Curriculum Development, InfoSecCD 2006, Kennesaw, Georgia, USA, September 22-23, 2006*, pages 85–91, 2006.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International*

*Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015a.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015b.

M. Iranpour and R. Safabakhsh. Reducing the embedding impact in steganography using hamiltonian paths and writing on wet paper. *Multimedia Tools Appl.*, 74 (17):6657–6670, 2015.

K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 2146–2153, 2009.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, Orlando, Florida, USA, Nov. 2014. ACM.

B. L. Kalman and S. C. Kwasny. Why tanh: choosing a sigmoidal function. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pages 578–581 vol.4, June 1992. doi: 10.1109/IJCNN.1992.227257.

A. D. Ker and T. Pevny. A Mishmash of Methods for Mitigating the Model Mismatch Mess. In *Media Watermarking, Security, and Forensics, Part of IS&T/SPIE 24th Annual Symposium on Electronic Imaging, SPIE'2014*, volume 9028, San Francisco, California, USA, 2014.

A. D. Ker, P. Bas, R. Böhme, R. Cogranne, S. Craver, T. Filler, J. J. Fridrich, and T. Pevný. Moving steganography and steganalysis from the laboratory into the real world. In *ACM Information Hiding and Multimedia Security Workshop, IH&MMSec '13, Montpellier, France, June 17-19, 2013*, pages 45–58, 2013.

A. Kerckhoffs. La Cryptographie Militaire. *Journal des Sciences Militaires*, IX, pp. 5-38 Jan. 1883, pp. 161-191, Feb. 1883.

G. Kipper. *Investigator's guide to steganography*. Auerbach publications, 2003.

J. Kodovský and J. J. Fridrich. On completeness of feature spaces in blind steganalysis. In *Proceedings of the 10th workshop on Multimedia & Security, MM&Sec 2008, Oxford, UK, September 22-23, 2008*, pages 123–132, 2008.

J. Kodovský and J. J. Fridrich. Steganalysis of JPEG images using rich models. In *Media Watermarking, Security, and Forensics 2012, Burlingame, CA, USA, January 22, 2012, Proceedings*, page 83030A, 2012.

J. Kodovsky, J. Fridrich, and V. Holub. On Dangers of Overtraining Steganography to Incomplete Cover Model. In *Proceedings of the Thirteenth ACM Multimedia Workshop on Multimedia and Security*, MM&MMSec'2011, pages 69–76, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0806-9. doi: 10.1145/2037252.2037266.

J. Kodovský, J. Fridrich, and V. Holub. Ensemble Classifiers for Steganalysis of Digital Media. *IEEE Transactions on Information Forensics and Security, TIFS*, 7(2):432–444, 2012.

J. Kodovskỳ, V. Sedighi, and J. J. Fridrich. Study of cover source mismatch in steganalysis and ways to mitigate its impact. In *Media Watermarking, Security, and Forensics*, page 90280J, 2014.

S. Kouider. *Adaptive Steganography : application to digital images in spatial domain*. Theses, Université Montpellier II - Sciences et Techniques du Languedoc, Dec. 2013. URL https://tel.archives-ouvertes.fr/tel-01020745.

S. Kouider, M. Chaumont, and W. Puech. Technical points about adaptive steganography by oracle (ASO). In *Proceedings of the 20th European Signal Processing Conference, EUSIPCO 2012, Bucharest, Romania, August 27-31, 2012*, pages 1703–1707, 2012.

S. Kouider, M. Chaumont, and W. Puech. Adaptive Steganography by Oracle (ASO). In *Proceedings of the IEEE International Conference on Multimedia and Expo, ICME'2013*, pages 1–6, July 2013. doi: 10.1109/ICME.2013.6607427.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA*, pages 2169–2178, 2006.

Y. Lecun. *A theoretical framework for back-propagation.* 1988.

Y. Lecun. *Generalization and network design strategies.* Elsevier, 1989.

Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Y. Lecun, L. Jackel, L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. *Learning algorithms for classification: A comparison on handwritten digit recognition*, pages 261–276. World Scientific, 1995.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *International Symposium on Circuits and Systems (ISCAS 2010), May 30 - June 2, 2010, Paris, France*, pages 253–256, 2010.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

D. Lerch-Hostalot and D. Megìas. Unsupervised Steganalysis Based on Artificial Training Sets. *Engineering Applications of Artificial Intelligence*, 50:45 – 59, 2016. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2015.12.013.

B. Li, M. Wang, J. Huang, and X. Li. A new cost function for spatial image steganography. In *2014 IEEE International Conference on Image Processing, ICIP 2014, Paris, France, October 27-30, 2014*, pages 4206–4210, 2014.

B. Li, W. Wei, A. Ferreira, and S. Tan. ReST-Net: Diverse Activation Modules and Parallel Subnets-Based CNN for Spatial Image Steganalysis. *IEEE Signal Processing Letters*, 25(5):650–654, May 2018. ISSN 1070-9908. doi: 10.1109/LSP.2018.2816569.

M. Lin, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations, ICLR 2014*, page 10, Banff, Canada, Apr. 2014.

Y. Y. Lu, Z. L. O. Yang, L. Zheng, and Y. Zhang. Importance of truncation activation in pre-processing for spatial and jpeg image steganalysis. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 689–693, Sep. 2019.

I. Lubenko and A. D. Ker. Steganalysis with mismatched covers: do simple classifiers help? In *Multimedia and Security Workshop, MM&Sec 2012, Coventry, United Kingdom, September 6-7, 2012*, pages 11–18, 2012a.

I. Lubenko and A. D. Ker. Going from small to large data in steganalysis. In *Media Watermarking, Security, and Forensics 2012, Burlingame, CA, USA, January 22, 2012, Proceedings*, page 83030M, 2012b.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602.

P. D. McDaniel, N. Papernot, and Z. B. Celik. Machine learning in adversarial settings. *IEEE Security & Privacy*, 14(3):68–72, 2016.

M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

R. M. Neal. Connectionist learning of belief networks. *Artif. Intell.*, 56(1):71–113, 1992.

A. R. Pach. Introduction to coding theory [book review]. *IEEE Communications Magazine*, 45(2):18, 2007.

J. Pasquet, S. Bringay, and M. Chaumont. Steganalysis with cover-source mismatch and a small learning database. In *22nd European Signal Processing Conference, EUSIPCO 2014, Lisbon, Portugal, September 1-5, 2014*, pages 2425–2429, 2014.

T. Pevný, P. Bas, and J. J. Fridrich. Steganalysis by subtractive pixel adjacency matrix. *IEEE Trans. Information Forensics and Security*, 5(2):215–224, 2010a.

T. Pevný, T. Filler, and P. Bas. Using high-dimensional image models to perform highly undetectable steganography. In *Information Hiding - 12th International Conference, IH 2010, Calgary, AB, Canada, June 28-30, 2010, Revised Selected Papers*, pages 161–177, 2010b.

B. Pfitzmann. Information hiding terminology - results of an informal plenary meeting and additional proposals. In *Information Hiding, First International Workshop, Cambridge, UK, May 30 - June 1, 1996, Proceedings*, pages 347–350, 1996.

L. Pibre, J. Pasquet, D. Ienco, and M. Chaumont. Deep Learning is a Good Steganalysis Tool When Embedding Key is Reused for Different Images, Even if There is a Cover Source-Mismatch. In *Proceedings of Media Watermarking, Security, and Forensics, MWSF'2016, Part of I&ST International Symposium*

*on Electronic Imaging, EI'2016*, pages 1–11, San Francisco, California, USA, Feb. 2016.

N. Provos and P. Honeyman. Detecting steganographic content on the internet. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2002, San Diego, California, USA*, 2002.

Y. Qian, J. Dong, W. Wang, and T. Tan. Deep Learning for Steganalysis via Convolutional Neural Networks. In *Proceedings of Media Watermarking, Security, and Forensics 2015, MWSF'2015, Part of IS&T/SPIE Annual Symposium on Electronic Imaging, SPIE'2015*, volume 9409, pages 94090J–94090J–10, San Francisco, California, USA, Feb. 2015.

Y. Qian, J. Dong, W. Wang, and T. Tan. Learning and transferring representations for image steganalysis using convolutional neural network. In *Proceedings of IEEE International Conference on Image Processing, ICIP'2016*, pages 2752–2756, Phoenix, Arizona, Sept. 2016.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention, MICCAI'2015*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

P. Sallee. Model-based steganography. In *IWDW*, 2003.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

D. Scherer, A. C. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks - ICANN 2010 - 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III*, pages 92–101, 2010.

P. Schöttle and R. Böhme. A Game-theoretic Approach to Content-adaptive Steganography. In M. Kirchner and D. Ghosal, editors, *Proceedings of the 14th International Conference on Information Hiding*, IH'12, pages 125–141. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-36372-6.

P. Schöttle and R. Böhme. Game theory and adaptive steganography. *IEEE Transactions on Information Forensics and Security*, 11(4):760–773, April 2016.

V. Sedighi, J. J. Fridrich, and R. Cogranne. Content-adaptive pentary steganography using the multivariate generalized gaussian cover model. In *Media Watermarking, Security, and Forensics 2015, San Francisco, CA, USA, February 9-11, 2015, Proceedings*, page 94090H, 2015.

V. Sedighi, R. Cogranne, and J. J. Fridrich. Content-adaptive steganography by minimizing statistical detectability. *IEEE Trans. Information Forensics and Security*, 11(2):221–234, 2016a.

V. Sedighi, J. J. Fridrich, and R. Cogranne. Toss that bossbase, alice! In *Media Watermarking, Security, and Forensics 2016, San Francisco, California, USA, February 14-18, 2016*, pages 1–9, 2016b.

T. Sharp. An implementation of key-based digital signal steganography. In *Information Hiding, 4th International Workshop, IHW 2001, Pittsburgh, PA, USA, April 25-27, 2001, Proceedings*, pages 13–26, 2001.

H. Shi, J. Dong, W. Wang, Y. Qian, and X. Zhang. SSGAN: secure steganography based on generative adversarial networks. In *Advances in Multimedia Information Processing - PCM 2017 - 18th Pacific-Rim Conference on Multimedia, Harbin, China, September 28-29, 2017, Revised Selected Papers, Part I*, pages 534–544, 2017.

Y. Q. Shi, C. Chen, and W. Chen. A markov process based approach to effective attacking JPEG steganography. In *Information Hiding, 8th International Workshop, IH 2006, Alexandria, VA, USA, July 10-12, 2006. Revised Selcted Papers*, pages 249–264, 2006.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

G. J. Simmons. The prisoners' problem and the subliminal channel. In *Advances in Cryptology*, pages 51–67. Springer, 1984.

X. Song, F. Liu, C. Yang, X. Luo, and Y. Zhang. Steganalysis of adaptive jpeg steganography using 2d gabor filters. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '15, pages 15–23, New York, NY, USA, 2015. ACM.

G. Stemmer, S. Steidl, E. Nöth, H. Niemann, and A. Batliner. Comparison and combination of confidence measures. In *Text, Speech and Dialogue, 5th International Conference, TSD 2002, Brno, Czech Republic September 9-12, 2002, Proceedings*, pages 181–188, 2002.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

T. Taburet, P. Bas, J. Fridrich, and W. Sawaya. Computing Dependencies between DCT Coefficients for Natural Steganography in JPEG Domain. In *Proceedings of the 7th ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec'2019*, pages 57–62, Paris, France, July 2019. ACM. doi: 10.1145/3335203.3335715.

S. Tan and B. Li. Stacked convolutional auto-encoders for steganalysis of digital images. In *Proceedings of Signal and Information Processing Association Annual Summit and Conference, APSIPA'2014*, pages 1–4, Siem Reap, Cambodia, Dec. 2014.

W. Tang, H. Li, W. Luo, and J. Huang. Adaptive steganalysis against WOW embedding algorithm. In *ACM Information Hiding and Multimedia Security Workshop, IH&MMSec '14, Salzburg, Austria, June 11-13, 2014*, pages 91–96, 2014.

W. Tang, S. Tan, B. Li, and J. Huang. Automatic Steganographic Distortion Learning Using a Generative Adversarial Network. *IEEE Signal Processing Letters*, 24(10):1547–1551, Oct 2017. ISSN 1070-9908. doi: 10.1109/LSP.2017.2745572.

W. Tang, B. Li, S. Tan, M. Barni, and J. Huang. CNN-based Adversarial Embedding for Image Steganography. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2019. ISSN 1556-6013. doi: 10.1109/TIFS.2019.2891237.

Y. Tang. Deep learning using linear support vector machines. In *In ICML*, 2013.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010. ISSN 1532-4435.

P. Wayner. Mimic functions. *Cryptologia*, 16(3):193–214, 1992.

P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

A. Westfeld. F5—a steganographic algorithm: High capacity despite better steganalysis. In *4th International Workshop on Information Hiding*, pages 289–302. Springer-Verlag, 2001.

A. Westfeld and A. Pfitzmann. Attacks on steganographic systems. In *Information Hiding, Third International Workshop, IH'99, Dresden, Germany, September 29 - October 1, 1999, Proceedings*, pages 61–76, 1999.

C. Xia, Q. Guan, X. Zhao, Z. Xu, and Y. Ma. Improving GFR Steganalysis Features by Using Gabor Symmetry and Weighted Histograms. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec'17, page 11, Drexel University in Philadelphia, PA, June 2017.

G. Xu. Deep Convolutional Neural Network to Detect J-UNIWARD. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec'17, page 6, Drexel University in Philadelphia, PA, June 2017.

G. Xu, H. Z. Wu, and Y. Q. Shi. Structural Design of Convolutional Neural Networks for Steganalysis. *IEEE Signal Processing Letters*, 23(5):708–712, May 2016a.

G. Xu, H.-Z. Wu, and Y. Q. Shi. Ensemble of CNNs for Steganalysis: An Empirical Study. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec'16, page 5, Vigo, Galicia, Spain, June 2016b. ISBN 978-1-4503-4290-2.

J. Yang, D. Ruan, J. Huang, X. Kang, and Y. Shi. An embedding cost learning framework using gan. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2019.

J. Yang, D. Ruan, X. Kang, and Y. Shi. Towards automatic embedding cost learning for JPEG steganography. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec 2019, Paris, France, July 3-5, 2019.*, pages 37–46, 2019.

J. Ye, J. Ni, and Y. Yi. Deep learning hierarchical representations for image steganalysis. *IEEE Transactions on Information Forensics and Security, TIFS*, 12(11):2545–2557, Nov. 2017.

M. Yedroudj, M. Chaumont, and F. Comby. How to Augment a Small Learning Set for Improving the Performances of a CNN-based Steganalyzer? In *Proceedings of Media Watermarking, Security, and Forensics, MWSF'2018, Part of I&ST International Symposium on Electronic Imaging, EI'2018*, Jan. 2018a.

M. Yedroudj, F. Comby, and M. Chaumont. Yedroudj-Net: An Efficient CNN for Spatial Steganalysis. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'2018, Calgary, AB, Canada, April 15-20, 2018*, pages 2092–2096, 2018b. doi: 10.1109/ICASSP.2018.8461438.

M. Yedroudj, F. Comby, and M. Chaumont. Steganography using a 3 player game. *CoRR*, abs/1907.06956, 2019. URL http://arxiv.org/abs/1907.06956.

M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

J. Zeng, S. Tan, B. Li, and J. Huang. Pre-training via fitting deep neural network to rich-model features extraction procedure and its effect on deep learning for steganalysis. In *Proceedings of Media Watermarking, Security, and Forensics 2017, MWSF'2017, Part of IS&T Symposium on Electronic Imaging, EI'2017*, page 6, Burlingame, California, USA, Jan. 2017a.

J. Zeng, S. Tan, B. Li, and J. Huang. Large-scale jpeg image steganalysis using hybrid deep-learning framework. *IEEE Transactions on Information Forensics and Security*, PP(99):1–1, Dec. 2017b. ISSN 1556-6013.

R. Zhang, F. Zhu, J. Liu, and G. Liu. Depth-Wise Separable Convolutions and Multi-Level Pooling for an Efficient Spatial CNN-based Steganalysis (previously named "efficient feature learning and multi-size image steganalysis based on cnn" on ArXiv). *IEEE Transactions on Information Forensics and Security, TIFS*, 2019. Under Submission.

B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

J. Zhu, R. Kaplan, J. Johnson, and L. Fei-Fei. HiDDeN: Hiding Data With Deep Networks. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Proceedings of the 15th European Conference on Computer Vision, ECCV'2018*, volume 11219 of *Lecture Notes in Computer Science*, pages 682–697. Springer, Sept. 2018.

# Résumé

La stéganographie d'image est l'art de la communication secrète dans le but d'échanger un message de manière furtive. La stéganalyse d'image a elle pour objectif de détecter la présence d'un message caché en recherchant les artefacts present dans l'image. Pendant une dizaine d'années, l'approche classique en stéganalyse a été d'utiliser un ensemble classifieur alimenté par des caractéristiques extraites "à la main". Au cours des dernières années, plusieurs études ont montré que les réseaux de neurones convolutionnels peuvent atteindre des performances supérieures à celles des approches conventionnelles d'apprentissage machine.

Le sujet de cette thèse traite des techniques d'apprentissage profond utilisées pour la stéganographie d'images et la stéganalyse dans le domaine spatial. La première contribution est un réseau de neurones convolutionnel rapide et efficace pour la stéganalyse, nommé Yedroudj-Net. Comparé aux méthodes modernes de steganalyse basées sur l'apprentissage profond, Yedroudj-Net permet d'obtenir des résultats de détection performants, mais prend également moins de temps à converger, ce qui permet l'utilisation des bases d'apprentissage de grandes dimmensions. De plus, Yedroudj-Net peut facilement être amélioré en ajoutant des compélements ou des modules bien connus. Parmi les amélioration possibles, nous avons evalué l'augmentation de la base de données d'entraînement, et l'utilisation d'un ensemble de CNN. Les deux modules complémentaires permettent d'améliorer les performances de notre réseau.

La deuxième contribution est l'application des techniques d'apprentissage profond à des fins stéganographiques i.e pour l'insertion. Parmi les techniques existantes, nous nous concentrons sur l'approche du "jeu-à-3-joueurs". Nous proposons un algorithme d'insertion qui apprend automatiquement à insérer un message secrètement. Le système de stéganographie que nous proposons est basé sur l'utilisation de réseaux adverses génératifs. L'entraînement de ce système stéganographique se fait à l'aide de trois réseaux de neurones qui se font concurrence : le stéganographeur, l'extracteur et le stéganalyseur. Pour le stéganalyseur nous utilisons Yedroudj-Net, pour sa petite taille, et le faite que son entraînement

ne nécessite pas l'utilisation d'astuces qui pourrait augmenter le temps de calcul. Cette deuxième contribution donne des premiers éléments de réflexion tout en donnant des résultats prometteurs, et pose ainssi les bases pour de futurs recherches.

**Mots clés:** Stéganographie, Stéganalyse, Apprentissage en profondeur, Réseaux de neurones convolutif, Réseaux antagonistes génératifs, Insertion strategique.

# ABSTRACT

Image steganography is the art of secret communication in order to exchange a secret message. On the other hand, image steganalysis attempts to detect the presence of a hidden message by searching artefacts within an image. For about ten years, the classic approach for steganalysis was to use an Ensemble Classifier fed by hand-crafted features. In recent years, studies have shown that well-designed convolutional neural networks (CNNs) can achieve superior performance compared to conventional machine-learning approaches.

The subject of this thesis deals with the use of deep learning techniques for image steganography and steganalysis in the spatial domain. The first contribution is a fast and very effective convolutional neural network for steganalysis, named Yedroudj-Net. Compared to modern deep learning-based steganalysis methods, Yedroudj-Net can achieve state-of-the-art detection results, but also takes less time to converge, allowing the use of a large training set. Moreover, Yedroudj-Net can easily be improved by using well-known add-ons. Among these add-ons, we have evaluated the data augmentation and the use of an ensemble of CNN; Both increase our CNN performances.

The second contribution is the application of deep learning techniques for steganography, i.e. the embedding. Among the existing techniques, we focus on the 3-player game approach. We propose an embedding algorithm that automatically learns how to hide a message secretly. Our proposed steganography system is based on the use of generative adversarial networks. The training of this steganographic system is conducted using three neural networks that compete against each other: the embedder, the extractor, and the steganalyzer. For the steganalyzer, we use Yedroudj-Net, this for its affordable size, and for the fact that its training does not require the use of any tricks that could increase the computational time. This second contribution defines a research direction by giving first reflection elements while giving promising first results.



**Keywords:** Steganography, Steganalysis, Deep Learning, Convolutional Neural Networks, Generative Adversarial Networks (GAN), Strategic embedding.