People’s Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

SAAD DAHLEB University, BLIDA 01

Faculty of Science

Computer Science Department

Master’s Thesis

Major: Security of Information Systems

Detection of Image Stegware Using Deep Learning

Author

TESTAS Dounia

Supervisor

Pr. BOUSTIA Narhimene

Acknowledgments

I would like to extend my sincerest thanks to my exceptional supervisor, Professor **BOUSTIA Narhimene**, her insightful guidance, unwavering support and patience have been crucial to the completion of this work.

Her dedication and belief in my abilities inspired me greatly and instilled in me the confidence to overcome the challenges and hardships I faced.

I am also deeply thankful to my family for their continuous love and encouragement throughout this journey, and to my dearest friends for being my pillars of support and a source of constant strength and motivation.

I am forever indebted to all those who have supported and assisted me, both near and far.

Abstract

In an era where privacy has become increasingly important with the constant informatisation of our day-to-day tasks, the quest to safeguard sensitive and personal information had led to the invention of various methods. Throughout history, the persistent need for secrecy and confidentiality has served as the driving force behind the development of these methods, including encryption techniques, anonymization protocols and secure communication systems. However, a paradoxical phenomenon has emerged as these very tools, which were initially intended to protect privacy, are now being exploited for the malicious purposes they were designed to guard against, one of these techniques is steganography.

The misuse of steganography to conceal malware within innocent media files, particularly images, has given rise to a significant cybersecurity concern known as stegomalware or stegware for short. Threat actors have recognized the potential of utilizing this technique to embed and distribute malicious payloads undetected. Consequently, traditional measures and defences are rendered powerless in the face of this sophisticated threat.

In this research, we aim to combine Deep Learning, Malware Analysis and Steganalysis techniques in order to put in place a system capable of dissecting and detecting stegware present specifically in PNG images. Our system comprises three main components. Firstly, we implement various steganalysis deep learning models proposed by researchers in the field, making the necessary adjustments and modifications to suit our case of study. The purpose of this first model is to determine the presence of steganography in images. Subsequently, we employ a module to extract hidden data from images identified as steganographic. Lastly, a text-based classification model is utilized to categorize the extracted data as either malicious or clean. The implementation details, rigorous testing, and comprehensive results will be discussed and presented in this study.

Keywords: Steganography, Malware, PNG Images, Deep Learning, Malware Analysis, Steganalysis, Detection, Classification.

Résumé

Au moment où la vie privée prend de plus en plus d'importance avec l'informatisation constante de nos tâches quotidiennes, la recherche de la protection des informations sensibles et personnelles a conduit à l'invention de différentes méthodes. Depuis toujours, la nécessité persistante du secret a été la force motrice derrière le développement de ces méthodes, notamment les techniques de cryptage, les protocoles d'anonymisation et les systèmes de communication sûrs. Cependant, un phénomène paradoxal a émergé, car ces outils, qui étaient à l'origine destinés à la protection de la vie privée, sont désormais exploités aux fins malveillantes contre lesquelles ils ont été conçus pour se protéger, l'une de ces techniques est la stéganographie.

Le mauvais usage de la stéganographie pour cacher des logiciels malveillants dans des fichiers multimédias innocents, en particulier les images, a donné lieu à un problème de cybersécurité appelé stegomalware ou stegware. Les attaquants ont reconnu la puissance d'utiliser cette technique pour éviter la détection en intégrant et en distribuant des payloads cachés dans des images apparemment inoffensives. En conséquence, les moyens de défense traditionnels sont impuissants face à cette menace sophistiquée.

Dans ce travail, notre but est de combiner l'apprentissage profond, l'analyse de logiciels malveillants et les techniques de stéganalyse pour créer un système capable de disséquer et de détecter les stegware. Notre système est constitué de trois éléments principaux. Tout d'abord, nous mettons en œuvre différents modèles d'apprentissage profond proposés par les chercheurs dans le domaine de stéganalyse, en effectuant les ajustements et les modifications nécessaires. Ce premier modèle vise à détecter la présence de la stéganographie dans les images. Nous utilisons ensuite un module pour extraire les données cachées de l'image identifiée comme stéganographique. Finalement, un modèle de classification sert à classer les données extraites comme étant malveillantes ou non est utilisé. Les détails de l'implémentation, les tests et les résultats complets seront discutés et présentés dans ces chapitres.

Mots-clés : Stéganographie, Logiciels malveillants, Images, Apprentissage en profond, Analyse de logiciels malveillants, Stéganalyse, Détection, Classification.

ملخص

في عصر تزايدت فيه أهمية الخصوصية مع الرقمنة المستمرة لنشاطاتنا اليومية، أدى السعي لحماية المعلومات الحساسة والشخصية إلى اختراع طرق متنوعة. على مر التاريخ، كانت الحاجة للسرية دافع مهم وراء تطوير هذه الطرق، بما في ذلك تقنيات التشفير وبروتوكولات التجهيز وأنظمة الاتصال الآمنة. ومع ذلك، ظهرت ظاهرة متناقضة، حيث يتم استغلال هذه الأدوات نفسها، التي كانت في الأصل مصممة لحماية الخصوصية، لأغراض خبيثة، ومن بين هذه التقنيات هي تقنية التورية او الستيغانوغرافيا.

سوء استخدام التسترغانوغرافيا لإخفاء البرامج الضارة في وسائط بريئة، مثل الصور، أدى إلى ظهور قلق أمني هام يعرف باسم ستيغومالوير أو ستيغووير بشكل مختصر. تعرفت الجهات المهددة للأمن على إمكانية استخدام هذه التقنية لتضمين وتوزيع الحمولات الضارة بدون ان يتم اكتشافها. وبالتالي، فان التدابير والدفاعات التقليدية أمام هذه التهديدات المتطورة لا مفعول لها.

في هذا البحث، نهدف إلى دمج تقنيات التعلم العميق وتحليل البرامج الضارة وتحليل الستيغانوغرافيا من أجل إنشاء نظام قادر على تحليل واكتشاف الستيغووير. يتكون نظامنا من ثلاث مكونات رئيسية. أولاً، نقوم ببرمجة نماذج تعلم عميق لتحليل الستيغانوغرافيا والمقترحة من قبل الباحثين في هذا المجال، مع إجراء التعديلات اللازمة. يهدف هذا النموذج الأول إلى التأكد من وجود الستيغانوغرافيا في الصور. بعد ذلك، نستخدم برنامج لاستخراج البيانات المخفية من الصور المحددة بأنها ستيغانوغرافية. وأخيرًا، يتم استخدام نموذج تصنيف نصي لتصنيف البيانات المستخرجة على أنها ضارة أو نظيفة. سيتم مناقشة تفاصيل التنفيذ وإجراء اختبارات دقيقة وتقديم النتائج الشاملة في هذه الدراسة.

الكلمات المفتاحية: الستيغانوغرافيا، البرمجيات الخبيثة، الصور، التعلم العميق، تحليل البرمجيات الخبيثة، التحليل المختبئ، الكشف، التصنيف.

Table of Content

List of Figures

List of Tables

List of Acronyms

ABS Absolute. p. 36

AES Advanced Encryption Standard. p. 39

ANN Artificial Neural Network. p. 27

AV AntiVirus. p. 22

BN Batch Normalization. p. 36

BOSS Break Our Steganographic System. p. 29, 30, 43

C&C Command and Control. p. 24

CNN Convolutional Neural Network. p. 28, 32-34, 36-38, 48-50

DCNN Denoising Convolutional Network. p. 38, 49

DCT Discrete Cosine Transform. p. 19, 36

DLL Dynamic Link Library. p. 23

DWT Discrete Wavelet Transform. p. 19

EK Exploit Kit. p. 24

FTP File Transform Protocol. p. 24

GIF Graphic Interchange Format. p. 24

GNCNN Gaussian Convolutional Neural Network. p. 29, 34-36, 48, 50

HPF High Pass Filter. p. 34, 36, 50

ICMP Internet Control Message Protocol. p. 17

IP Internet Protocol. p. 17

JPEG Joint Photographic Experts Group. p. 29, 36, 48

JS JavaScript. p. 24

LSTM Long Short-Term Memory. p. 39

LSB Least Significant Bit. p. 18-20, 45

LoG Laplacian of Gaussian. p. 34

MaaS Malware as a Service. p. 23

ML Machine Learning. p. 42

MSE Mean Squared Error. p. 46

OSI Open System Interconnection. p. 17

PE Portable Executable. p. 23

PNG Portable Network Graphics. p. 24, 29, 31, 40, 42-44, 48-52

PSNR Peak Signal-to-Noise Ratio. p. 46, 47

PVD Pixel Value Differencing. p. 19

RAT Remote Access Trojan. p. 23

RGB Red Green Blue. p. 33, 42, 43

RNN Recurrent Neural Network. p. 32, 39

SMTP Simple Mail Transform Protocol. p. 24

SSIM Structural Similarity Index. p. 46, 47

TCP Transmission Control Protocol. p. 17

UDP User Datagram Protocol. p. 17

URL Uniform Resource Locator. p. 23, 24

VBS Visual Basic Scripting. p. 23

General Introduction

The necessity for secrecy and discretion has existed as long as the need for communication between individuals and society. Throughout history, people have always sought ways to ensure secure and confidential transmission of sensitive information. In ancient civilizations, for example, King Histiaeus employed a unique method: he shaved the head of a slave, tattooed a secret message on his scalp, waited for his hair to grow back, and then sent him to the Greeks [1]. Fast forward to World War II, where steganography was used in the form of Microdots by military troops to pass messages via insecure postal channels. Agents used tiny photographs or documents reduced to the size of a dot. These dots were then placed on seemingly harmless-looking objects, such as the bottom of clothes, postage stamps, or letters [2].

Now back to the Digital Era, new applications for steganography have been found in digital media. For example, digital images can nowadays be used as carriers to conceal all sorts of data using different embedding techniques that varies in levels of stealthiness. These new found ways didn't go unnoticed by the web criminals, who are always in search for different manners to pass under the radar of detection methods.

The incorporation of steganography by existing malware families introduces an additional layer of threat and danger, augmenting the already menacing nature of malware in its conventional and normal form. By concealing malicious payloads within PNG digital images, cybercriminals can exploit unsuspecting individuals and organizations, and go completely unnoticed. This is because antivirus software and other existing security measures are not specifically designed to combat this type of malware. As a result, there is a pressing need to strengthen our efforts to neutralize and mitigate the dangers associated with this new wave of malware.

In our pursuit to create a safer system, we fuse multiple disciplines and work to extend the pre-existing approaches, expanding their applicability to encompass larger more diverse and realistic scenarios. In the first chapter, we provide an introduction to the fundamental concepts of steganography, steganalysis, malware, and malware analysis. We then present a brief overview of dep learning techniques, specially focusing on CNNs and autoencoders. Moving forward, we lay out our proposed approach, discussing the various implementations. Finally, in our fourth chapter we discuss the conducted tests and analyse the results obtained and the obstacles encountered.

Chapter 01

Steganography and Stegomalware

1.1 Introduction

Steganography is considered one of the key components in facilitating the process of hiding malicious programs to ensure their undetection by protective measures. It serves as the primary method utilized by malicious actors to ensure the successful infiltration of their malware into victims' machines. In this chapter, we will cover fundamental concepts of steganography and its related techniques. Additionally, we will provide a brief introduction to the concept of malware, along with examples of families that have already adopted this technique.

1.2 Steganography

The primary motive behind concealing information is to keep it away from the eyes of those not intended to see it. Steganography, as a technique to achieve this, derives its name from the Greek term "steganos," meaning "secret," combined with "graphy" signifying "writing." In its simplest form, steganography involves hiding information, whether it be data concealed within a digital file, an image masked by another image, or words written in invisible ink [3]. Although steganography is often confused with cryptography and watermarking due to their shared objectives, they remain fundamentally distinct methods.

1.2.1 Key Terminology

In the context of steganography, certain terms are used to refer to important actors involved in the process. The following is a list of key terms and their meanings:

Stego files: Also known as carriers, these files contain embedded hidden information resulting from the application of a steganographic technique.

Cover files: These files have the potential to be used as carriers, meaning they can be used to conceal hidden information through an embedding method that supports them. Cover files can include any file type as long as there is a compatible embedding technique available.

Clean files: In contrast, clean files are untouched files that have never undergone any modifications using steganography. These files remain unaltered and free from any hidden information.

Embedding rate: It refers to the quantity of hidden information that can be concealed within the carrier file without affecting its perceptual quality or arousing suspicion. It is typically measured in bits or bytes.

1.2.2 Digital steganography

While steganography has ancient origins rooted in early history, it has evolved and expanded significantly with the emergence of technology, presenting new opportunities for its implementation. In contemporary practice, digital steganography is defined as “The art and science of hiding information into covert channels, so as to conceal the information and prevent the detection of the hidden message.” By Shih, Frank in his book [4]. To enhance effectiveness, steganography is often combined with modern cryptographic techniques, adding an extra layer of confidentiality and security.

1.2.3 Cryptography and watermarking

Information security is a critical requirement that can be attained through various means. In general, information security systems can be categorized into two classes: encryption and information hiding [5], as illustrated in Figure **1.1**. Steganography, watermarking, and cryptography all strive to safeguard information using distinct approaches.

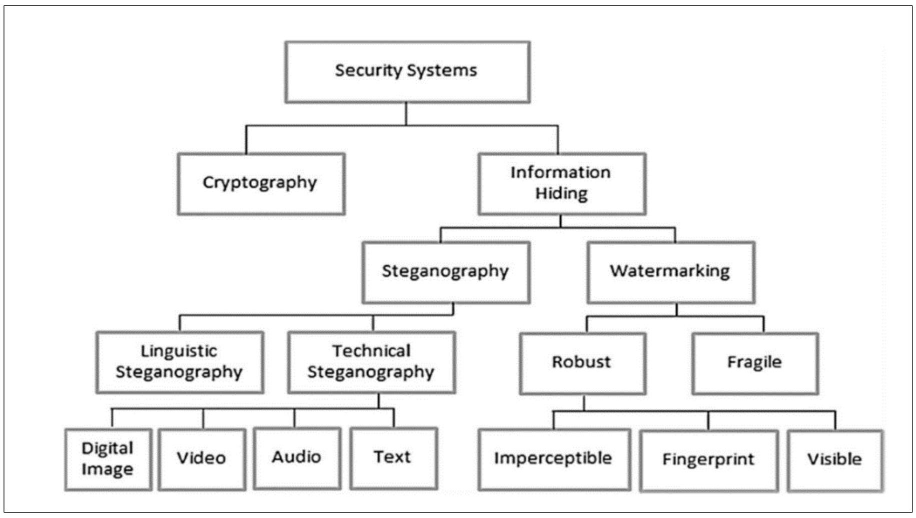


Figure 1.1 Classification tree of a general data security system. [6]

1.2.3.1 Watermarking

One approach to ensure the authenticity of information is through watermarking. An electronic watermark serves as an imprint on a digital file, providing evidence of its originality and reducing the risk of counterfeiting [3]. In the context of hiding confidential data within different media files, watermarking and steganography share common objectives. They both possess attributes such as data capacity, security, imperceptibility, and robustness [7].

1.2.3.2 Cryptography

The key principle of cryptography is to make information or data illegible to ensure its confidentiality, making it incomprehensible to unauthorized individuals. Cryptography is commonly used in securely transmitting data through insecure channels like the internet safeguarding it from unauthorized access. In cryptographic terms, the original information is referred to as "plaintext," and the process of transforming it is known as "encryption," resulting in "ciphertext." Encryption involves the use of specific algorithms called "encryption algorithms" and requires an "encryption key" as input. To retrieve the information, the recipient employs a "decryption algorithm" along with the corresponding "decryption key" [8].

The integration of cryptography principles into steganography leads to the following classification.

Pure steganography

Pure steganography is when data is hidden within an object without using any encryption keys. Essentially, the data is embedded as is, without being encrypted first. However, this approach is not very secure [9]. If someone unauthorized manages to figure out the specific embedding technique being used, they can easily extract the hidden data.

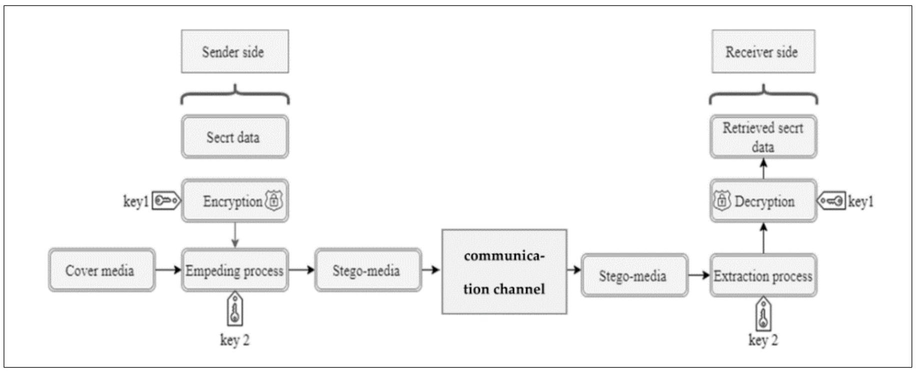


Figure 1.2 Block diagram of a steganographic system [6].

Secret key steganography

Similar to pure steganography, secret key steganography uses an embedding algorithm to conceal data within a selected digital carrier. However, unlike pure steganography, it incorporates the use of a symmetric key to encrypt and decrypt the data prior to embedding and after extraction [9]

Public key steganography

On the other hand, public key steganography employs an asymmetric key to encrypt and decrypt the data before and after transmission [9]. Public key works by using a pair of mathematically related keys: a public key and a private key. The public key is freely distributed, allowing anyone to encrypt data using this key, while the private key is kept secret and is used for decrypting the encrypted data The detailed process is illustrated in Figure **1.2**.

1.2.4 Types of modern steganography

Taking into consideration the type of digital carrier and the format of the embedded data, steganography can be classified into five main types, as depicted in Figure **1.3**:

1.2.4.1 Text steganography

Text steganography involves concealing information within text files through various techniques. These techniques may include modifying the formatting of existing text, altering individual words, generating random strings, and constructing coherent text using context-free grammars [10].

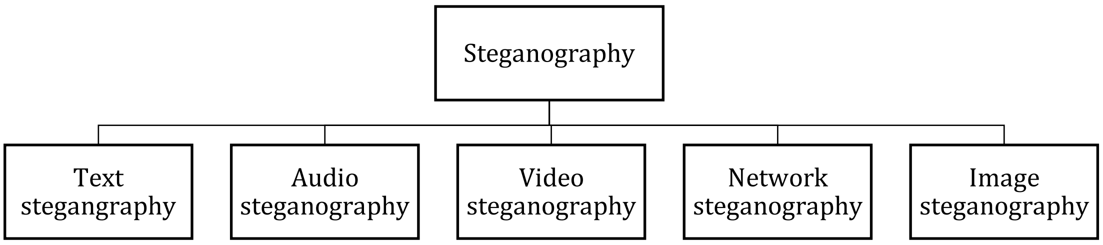


Figure 1.3: The different types of Steganography

1.2.4.2 Audio steganography

In audio steganography, covert messages are embedded within audio signals by modifying the binary sequences of accompanying audio files. This type of steganography presents a greater challenge in concealing secret messages using digital sound [10].

1.2.4.3 Video steganography

Digital video format provides the capability for concealing information through video steganography. This method offers the advantage of fitting a substantial amount of hidden data within a dynamic stream of images and sounds. It can be seen as a combination of audio and visual steganography, blending elements from both domains [11].

1.2.4.4 Network steganography

It is a technique that involves embedding data into network control protocols like TCP, UDP, and ICMP, which are used for transporting data. Within the OSI model, there are hidden communication channels that can be utilized in conjunction with steganography [5]. For instance, you can conceal information within certain optional header fields of TCP/IP packets.

1.2.4.5 Image steganography

In image steganography, data is concealed by using an image as the cover object. Images are commonly used in digital steganography due to their high bit depth [12]. An image is typically represented as an N * M matrix in memory, where each entry corresponds to the intensity value of a pixel. During the process of embedding a message into the image, specific pixels are selected and their values are modified according to an encryption algorithm.

1.2.5 Image steganography techniques

It is important to highlight the significant role that compression plays in determining the effectiveness of steganographic algorithms. While lossy compression methods reduce image file sizes, they also increase the possibility of partial loss of embedded messages due to the removal of image data. On the other hand, lossless compression techniques do not compress image files as much, ensuring minimal loss of embedded information [12].

To resolve this issue, researchers have come up with various steganographic algorithms including the following:

1.2.5.1 Spatial domain steganography

The spatial domain refers to the direct manipulation of the pixel values and their positions in an image without any transformation. It involves working with the original pixel grid to perform different operations.

Least significant bit (LSB)

An image is a visual representation composed of individual pixels, where each pixel represents a specific element of the image. It is a collection of small units that together form the visual content. Each pixel of an image consists of three bytes representing the intensity of the primary colours (RGB), as shown in Figure **1.4**.

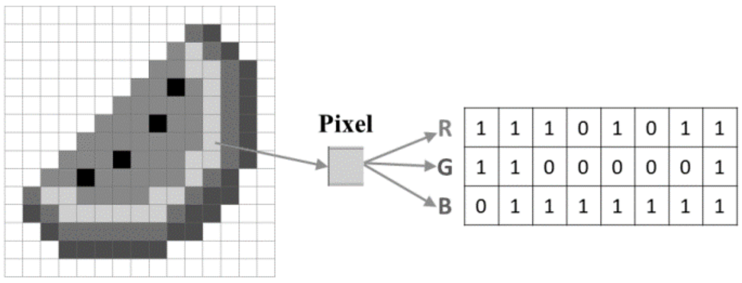


Figure 1.4 Pixel size of different colour. [10]

In the LSB method, the least significant bit of each pixel in the image is utilized to perform an Exclusive OR (XOR) operation with secret data. This process ensures that the least significant bit values of the pixels store the secret data [13].

An example [14] of this is the insertion of letter “D” in a 24-bit image, we know the binary representation of letter “D” (ASCII value of 68) is 01000100.

After the embedding in the LSBs, we obtain the following results:

Pixel 1: (0010011**1** 1110100**1** 1100100**0**) ⇒ (0010011**1** 1110100**0** 1100100**0**)

Pixel 2: (0010011**1** 1100100**0** 1110100**1**) ⇒ (0010011**1** 1100100**0** 1110100**0**)

Pixel 3: (1100100**0** 0010011**1** 1110100**1**) ⇒ (1100100**0** 0010011**1** 1110100**1**)

Pixel value differencing

The PVD-based steganographic scheme is an edge adaptive method where the number of embedded bits depends on the variation between a pixel and the pixels surrounding it [15], The basic idea behind PVD is to calculate the difference between the pixel value of a selected pixel and the pixel values of its neighbouring pixels. The larger the difference between the pixel and its neighbours, the greater the capacity to embed message bits.

1.2.5.2 Transform domain steganography

On the other hand, the transform domain refers to a specific representation of data obtained by applying a mathematical transform to the original image. It involves converting the image from its original spatial domain into a different domain, such as frequency or wavelet domain, using the following techniques:

Discrete Cosine transform (DCT)

DCT is a technique commonly used in transform domain steganography, particularly for lossy image formats. It allows the transformation of an image from the spatial domain to the frequency domain. the lossy image is divided into components based on their frequency importance, namely low frequency, middle frequency, and high frequency components [16]. The essential visual elements are preserved in the low frequency components, while the secret information is embedded by modifying the coefficients of the middle frequency components without significantly affecting the visibility of the image.

Discrete Wavelet transform (DWT)

DWT steganography is another technique also used for lossy images. It has been introduced as a highly flexible and efficient method for processing signals. DWT allows the concentration of signal energy into wavelet coefficients, enabling more efficient storage compared to blocks of pixels [17]. With wavelets, an image can be converted into a series of wavelet coefficients that can be stored in a more efficient manner.

1.3 Steganalysis

As steganography gained popularity, the need for a method to counter it, arose under the name Steganalysis. Steganalysis is the art and science of detecting concealed messages embedded in images using steganography. Its purpose is to determine if undetectable messages are present, allowing the steganographer to detect, extract, disable, or modify the messages before reaching the intended recipient.

1.3.1 Types of steganalysis

Steganalysis plays a significant role in selecting distinguishing features that might be shown by Stego- and Cover-objects. Two types of features are commonly observed: deep features and handcrafted features, often referred to as "statistical features" or "specific features". Steganalysis approaches based on these features can be classified into the following categories:

1.3.1.1 Signature steganalysis

In this approach, features are treated as distinct patterns or signatures. If the steganographic embedding technique is known, it becomes easier to identify and extract recurring special patterns, such as histogram arrangements, intensity ranges, and more. This type is referred to as "target" or "specific" steganalysis.

Conversely, the "universal" type considers features as behavioural patterns regardless of the embedding technique. Some steganography methods follow a sequential or linear access approach when embedding, which can create noticeable patterns [18].

1.3.1.2 Statistical steganalysis

Statistical steganalysis primarily depends on extracting statistical features and properties from cover- and stego-images. Similar to the previous type, it includes both “universal” and “target” methods. Target techniques are developed by studying and analysing specific steganographic embedding techniques to identify modified statistical features resulting from the embedding process. Deep understanding of the embedding techniques enhances steganalysis accuracy and gives rise to various categories based on the embedding domain (such as LSB matching steganalysis, LSB embedding steganalysis, Transform domain steganography steganalysis, etc.) [18].

Conversely, universal statistical steganalysis does not target specific steganography techniques. It employs learning and training concepts to identify sensitive statistical features with distinguishing capabilities. These features are utilized to construct learning models for machine learning and neural networks [19].

1.3.1.3 Deep steganalysis

We refer to this category as deep steganalysis, named after the concept of deep features. Neural networks have gained popularity in both deep learning and classification tasks due to their accuracy and ability to enable profound understanding for improved robustness and effectiveness in semantic representation. Deep steganalysis shares similarities with universal statistical steganalysis, as it does not rely on specific embedding steganography techniques [21]. However, the distinction lies in the extraction of deep features in contrast to hand-crafted features.

1.4 Stegomalware

In simple terms, Stegomalware or Stegware is a type of malware that utilizes steganography to conceal its malicious payload within an image [22]. An instance of this is when a script is embedded within an image, where no suspicious elements are apparent upon viewing. However, the payload is cleverly manipulated so that when the image file is executed as a script, the appended malicious code seamlessly executes as well. In such a scenario, the downloaded image file evades detection as it appears harmless and is executed by the web page without triggering any defence measures. The hidden script then gains access to another image containing the primary attack payload.

1.4.1 Malicious software or Malware

Various terms, such as malicious code, malcode, and malware, are used to describe malicious software. Different definitions have been proposed, and for the purpose of this context, we adopt the definition provided by Vasudevan and Yerraballi [23], which defines malware as "a generic term that encompasses viruses, trojans, spyware, and other intrusive code." Some of which are shown in figure **1.5**.

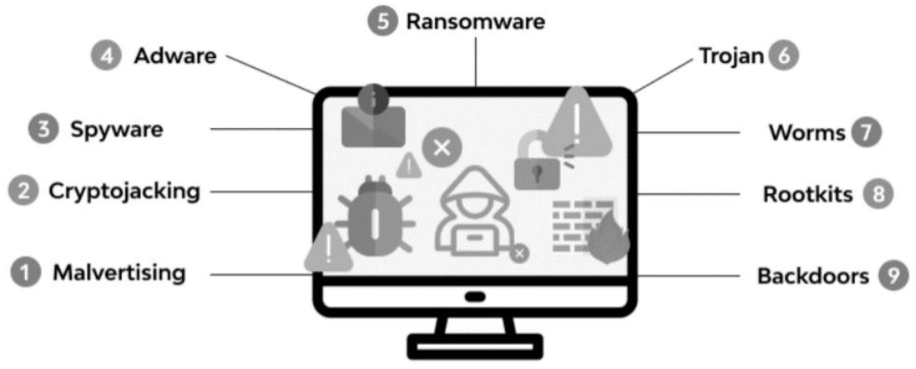


Figure 1.5 Common types of malware [20]

1.4.2 Malware analysis

Over the past decade, malware analysis and detection techniques have undergone significant evolution in response to the development of various malware techniques aiming to escape being detected by security measures. The rapid growth of diverse malware forms has posed considerable challenges for forensic investigators, making it increasingly difficult to provide timely responses. As a result, the integration of Machine Learning (ML) into malware analysis has become imperative, enabling automation of various aspects of static and dynamic malware investigation.

1.4.2.1 Dynamic malware analysis

Dynamic malware analysis involves observing the behaviour of malware in a controlled environment, such as a virtual machine or emulator. It allows for analysing the actual behaviour during runtime, bypassing the limitations of static analysis. By executing the malware in a restricted environment, it is possible to monitor its actions, including changes to registry keys and privileged access to the operating system [24]. This approach provides advantages such as detecting known and unknown malware, including obfuscated and polymorphic variants. However, dynamic analysis can be resource-intensive and time-consuming, and it may suffer from incomplete code coverage and potential risks to third-party systems. Despite these challenges, dynamic analysis is valuable for understanding and countering malware threats.

1.4.2.2 Static malware analysis

Static analysis on the other hand, involves analysing executable files without executing them in a controlled environment. It focuses on the structure and static attributes of the files, extracting information without running them. Malware often employs binary packers to avoid analysis, requiring the files to be unpacked and decompressed before analysis. Disassembler tools can be used to decompile Windows executable files and extract patterns to identify attackers. Static analysis is conducted manually and can be challenging due to the loss of information during compilation [25]. However, it provides valuable insights into the structure and characteristics of malware without the need for execution.

Some of the techniques employed in static malware analysis include: checking File format, AV scanning, Packer detection and Disassembly.

Disassembly:

Static analysis predominantly involves the disassembly of a provided binary. This process utilizes tools that can reverse the machine code into assembly language, such as IDA Pro. By examining the reconstructed assembly code, analysts can investigate the program logic and discern its underlying intent.

1.4.3 Stegomalware examples

We present 3 examples of malware families utilizing steganography and explain how they work:

1.4.3.1 Ransomware Cerber

A ransomware is generally defined as "a kind of malware which demands a payment in exchange for a stolen functionality" [26]. By running an executable, the victim's machine data is encrypted, the adversary then demands a ransom in exchange for a decryption key.

Cerber is a variation named after Cerberus, a 3-headed dog guiding the entrance to Hades in Greek Mythology. The Cerber ransomware attack starts with a decoy document that contains malicious macro code. When the document is opened, it drops a VBScript file with a random name in the user's "%APPDATA%" directory. This VBScript file is executed using the "wscript.exe" process, which downloads an image file named "mhtr.jpg" from specific URLs. The image file appears benign and displays content related to "zen-coding," but it contains hidden malware embedded using steganography [27]. This allows the transmission of executables without raising suspicion from network monitoring devices.

1.4.3.2 RAT Agent Tesla

RATs, short for Remote Access Trojans, as the name suggests, are a type of Trojan malware. While a Trojan is not classified as a virus, it may potentially harbour a virus within it and deceive users by appearing as something beneficial. RAT is also a type of MaaS "Malware-as-a-Service", it allows threat actors to gain control of the system and access the victim's information by creating a backdoor in the user's system.

To summarize briefly, Agent Tesla is a Remote Access Trojan (RAT) that is usually delivered through phishing emails and uses various evasion techniques to avoid detection and analysis.

Agent Tesla's main functionalities include keylogging, screen capturing, form-grabbing, and credential theft. It targets popular software programs like Google Chrome, Mozilla Firefox, and Microsoft Outlook to extract sensitive information.

Steganography plays its role in storing a PE file in a bitmap image, to be extracted by the first stage DLL module, then the data is collected from this image in the main payload, decrypting the collected data, and generating the second stage module. This second stage DLL module is heavily obfuscated to complicate analysis.

Once the second stage DLL is loaded into memory, it performs further decryption routines to obtain the final payload. After the final payload is decrypted, the malware injects its code into the main process and starts stealing computer information, including browser data, keystrokes, clipboard data, FTP credentials, and more [28].

Agent Tesla encrypts stolen data before communicating with its command and control (C&C) server and uses the TOR client to maintain anonymity. Stolen data is exfiltrated over SMTP (Simple Mail Transfer Protocol), and for persistence, the malware drops its copy at a specific location and creates a run entry.

1.4.3.3 Astrum Exploit Kit

The Astrum Exploit Kit, also known as Stegano, was discovered in 2016 during the AdGholas Malvertising Campaign. This campaign was launched by the cybercriminal group known as AdGholas from 2015 to 2017 and targeted a large number of websites. The Astrum EK is an image-based exploit kit used to distribute various malicious payloads such as backdoors, trojans, spyware, and ransomware using steganography [29].

The main target of Astrum EK is users with unpatched Windows systems who are infected through poorly configured third-party webservers.

1. When a user clicks on the malicious advertisement, the index.html file of their browser loads a JavaScript file. This file contains obfuscated malicious code and reports the victim's local environment back to the server.
2. Based on the environment information received, the server responds with an advertisement banner, a steganographic PNG image that contains hidden JavaScript code.
3. The hidden JavaScript code attempts to further analyse the browser and computer environment, focusing on detecting packet capture, sandboxing, virtualization software, security products, and drivers.
4. If no signs of monitoring are detected, the victim is redirected to the landing page of the Stegano exploit kit via the TinyURL service. The landing page loads a Flash file.
5. The Flash file invokes a JS code which returns a shell code containing the URL of the payload and a password. This shell code collects information about installed security products and, if the results are favourable, downloads the encrypted payload disguised as a GIF image.
6. The payload is then decrypted and launched using regsvr32.exe or rundll32.exe.

1.5 Conclusion

In this chapter, we present the key concepts to understand how steganography and malware converge to form stegomalware. In the next chapter, we will delve into deep learning techniques that can aid in countering this emerging threat, along with prominent works in this field.

Chapter 02

Artificial Neural Networks

2.1 Introduction

Deep learning plays a vital role in malware detection by leveraging extensive training data to accurately classify and identify malicious software. In steganalysis, deep learning models facilitates the detection of hidden information in digital media by learning statistical characteristics and subtle changes associated with steganographic content. This chapter introduces a brief overview of artificial neural networks (ANNs). It explores ANNs, including CNNs, autoencoders and RNNs. We'll also cover applications such as denoising.

2.2 Artificial Neural Networks

An artificial neural network can be formally defined as a powerful computational model that consists of interconnected processing units. These units have the ability to store and utilize experiential knowledge in a parallel and distributed manner [30].

2.2.1 Neural network structure

A basic artificial neural network typically consists of three key components: an input layer, one or more hidden layers composed of interconnected neurons, and an output layer as depicted in figure **2.1**. Neurons, inspired by biological neurons, are the fundamental processing units within the network. Each neuron takes inputs, applies weights to them, and passes the result through an activation function.

Input layer: this layer is responsible for receiving the input data, which could be numerical values, images, or any other form of structured data. Each input node in the input layer represents a feature or attribute of the data.

Hidden layer: situated between the input and output layers, each neuron composing this layer takes inputs, applies weights to them, and passes the result through an activation function. The goal is to enable the network to learn and extract relevant features from the input data.

Output layer: the role of this layer is to produce predictions based on the processed information from the hidden layers. The number of neurons in the output layer depends on the type of problem the ANN is designed to solve.

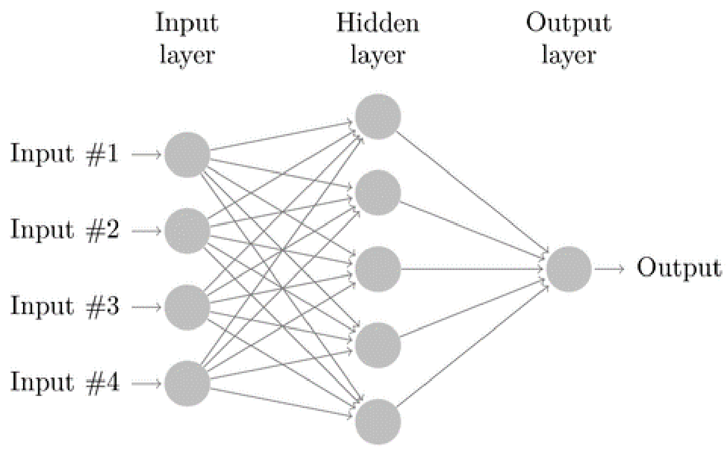


Figure 2.1: Basic Neural Network Structure [31]

2.3 Types of neural networks

Making certain modifications to the basic ANN structure gives birth to new types and architectures. These modifications include changing connectivity patterns, activation functions and learning algorithms.

2.3.1 Recurrent neural networks

RNNs are a more advanced structure in which neurons within a layer are interconnected and allow for feedback, resulting in information flowing in cycles [32] an example of that would be the illustration in figure **2.2**. This unique architecture makes RNNs better suited for tasks such as natural language processing (NLP) and speech recognition, given their effectiveness in processing sequential and temporal data.

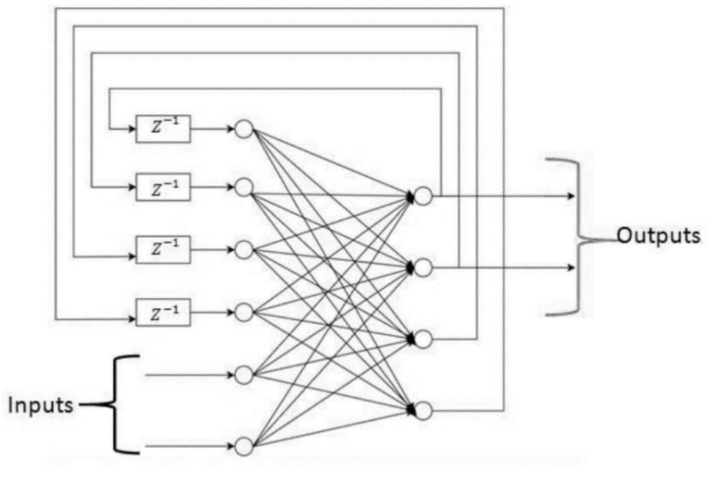


Figure 2.2: Recurrent network with hidden neurons. [32]

2.3.2 Convolutional neural networks

A Convolutional Neural Network (CNN) is a popular algorithm widely used in the field of deep learning, especially for image-related tasks. It consists of input, convolutional, pooling, fully connected, and output layers as shown in figure **2.3**. CNNs automatically identify relevant features without human supervision [33], allowing them to extract meaningful patterns from images. Convolutional layers detect local patterns, pooling layers reduce spatial dimensions, and fully connected layers perform high-level reasoning. The output layer provides final classification or regression results. Overall, CNNs have revolutionized image analysis and recognition tasks, playing a crucial role in computer vision applications.

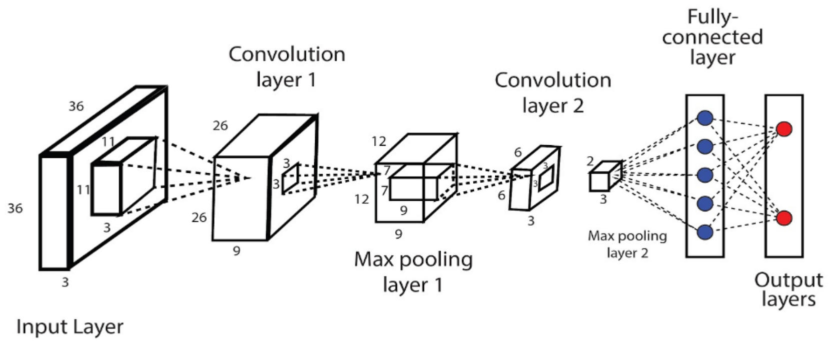


Figure 2.3: Basic CNN structure [34]

2.3.3 Autoencoders

Autoencoders were initially introduced as a neural network architecture aimed at reconstructing its input data, as described in [35]. They serve as a powerful tool for unsupervised learning, focusing on obtaining a compressed and meaningful representation of the input data. By learning to encode and decode the data, autoencoders aim to capture the essential features and patterns within the dataset, that is clearly described in figure **2.4**. This "informative" representation obtained by autoencoders can be utilized for numerous purposes, namely, image denoising anomaly detection.

Encoder: the encoder component of the network is responsible for compressing or encoding the input data into a representation in a latent space. This compressed representation often appears distorted or unintelligible compared to the original data.

Decoder: the decoder component is responsible for decoding or reconstructing the encoded data from the latent space back to its original dimensions. However, the reconstructed data is typically a lossy approximation of the original data, meaning that it may not perfectly match the exact details of the original input.

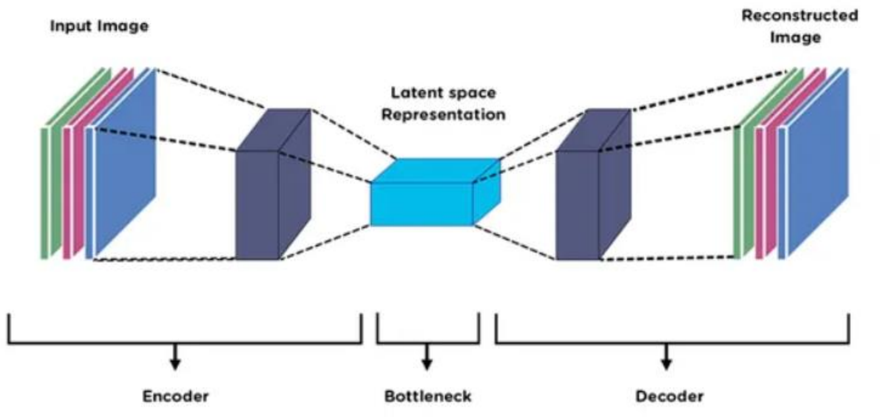


Figure 2.4: Basic structure of an Autoencoder [36]

2.4 Related Work

To the best of our knowledge, no previous research has been conducted on the identification and detection of Stegware in PNG images. Therefore, considering the significant role of steganalysis in the detection of hidden Stegware, we have decided to highlight four existing works in the field of steganalysis, ordered in table **2.1.** These works could be adapted to work with PNG images through appropriate adjustments and modifications.

Authors	Year	Dataset	Technique	Performance	Evaluation metrics
Qian et al [39].	2015	BOSSbase 1.01, ImageNet	GNCNN	**0.3bpp:** HUGO: 0.338, WOW: 0.343, S-UNIWARD: 0.359	Detection error
Xu et al [40].	2016	BOSSbase 1.01	Xu-Net	**0.4bpp:** S-UNIWARD:79.53, HILL:75.47	Accuracy
Boroumand et al [37].	2019	BOSSbase and BOWS2	SRNet	**0.2bpp:** Spatial (S-UNIWARD: 0.2090, HILL: 0.2353, WOW: 0.1676); QF75, JPEG (J-UNIWARD: 0.1889, UED-JC -.0568).	Detection Error

Singh et al [38].	2021	BOSSbase	SFNet	**0.2bpp:** WOW:0.1579, S-UNIWARD:0.1964, HILL: 0.2438	Detection Error

Table 2.1: Related works in Steganalysis and Stegware detection

2.5 Conclusion

In this chapter, we have introduced the fundamental techniques of deep learning models that are extensively utilized in addressing the problem at hand. Building upon these works and foundational knowledge, we will proceed to construct our proposed system, which will be discussed in the following chapter.

Chapter 03

Proposed Approach

3.1 Introduction

Due to the fact that stegomalware detection combines three separated research fields, as mentioned previously, that are vast on their own, few studies have been conducted to address this problem. In the following chapter, we'll be presenting our proposed approach to tackle both steganalysis and malware detection aspects of this solution. Additionally, we will discuss various implementations proposed by researchers for steganalysis generalised to work for PNG images.

3.2 Methodology

The primary principle of our system is to decompose the stegware into distinct components and tackle each component using its corresponding counterattack method, as illustrated in Figure **3.1**. By leveraging the advantages of steganalysis and malware detection within a deep learning framework, our proposed system is comprised of three components:

1. **Steganalyzer:** The first part of our system involves implementing 4 different models to select the one with the best performance. These models serve as steganalyzers, detecting the presence of steganography in an image, regardless of whether the embedded data is harmless or malicious.
2. **Data Extraction Module:** The second part of our system is a data extraction module. This module, implemented as a Python script, is responsible for extracting the embedded data from the detected steganographic images. To accomplish this, we have chosen to use the Openstego Extractor tool, which was also employed during the preparation of the dataset.
3. **Text Classifier:** The extracted data is then fed into the second deep learning model, a simple classifier designed to classify whether the text data is benign or malicious. This classifier serves as the third part of our system and aids in determining the nature of the extracted information.

Please note that Figure **3.2** provides a more detailed visual representation of the system architecture, illustrating the flow of data and the interaction between the components.

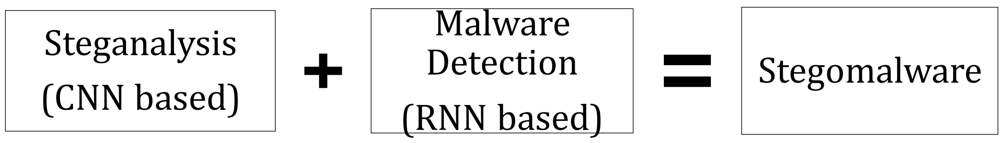


Figure 3.1 : The proposed approach

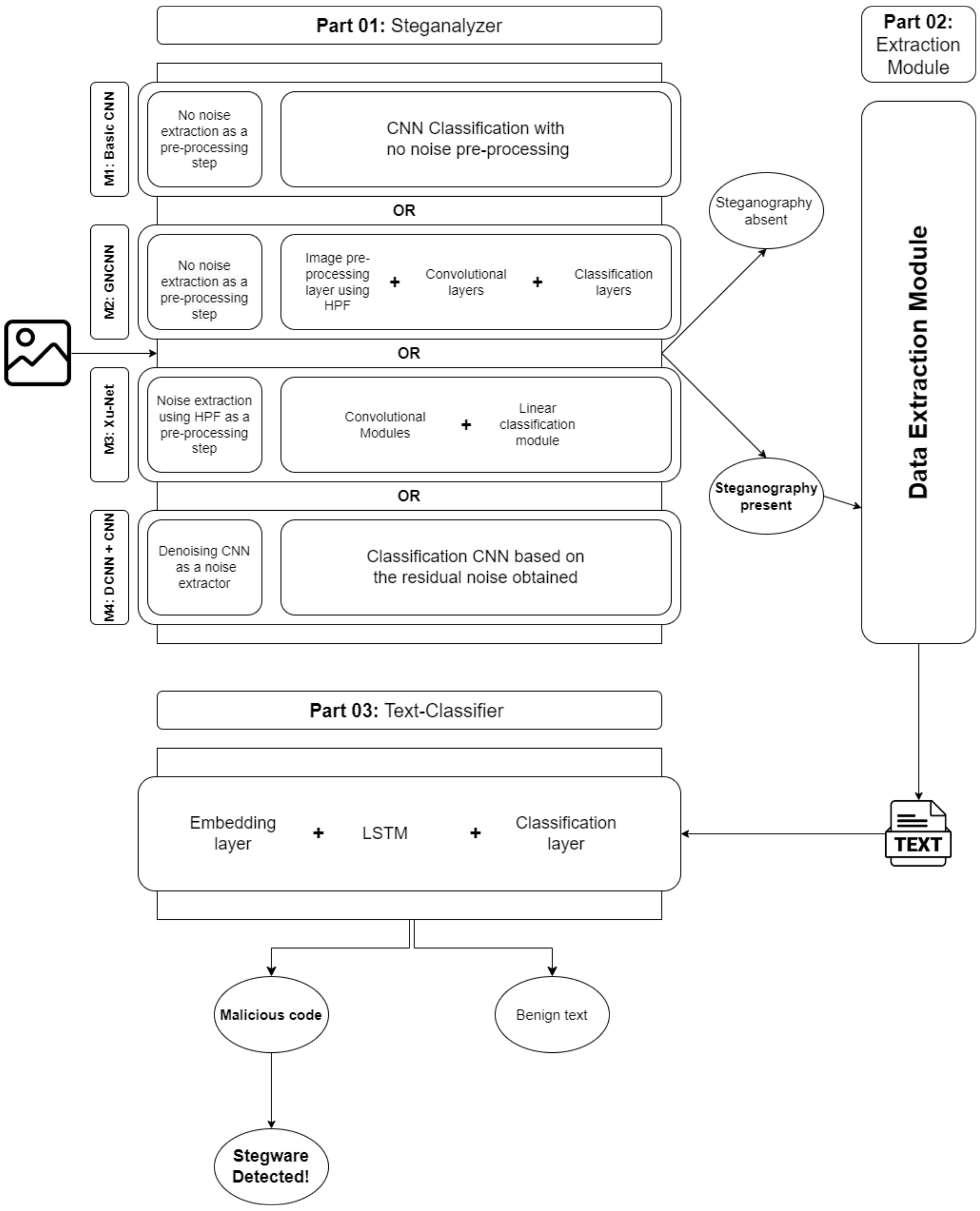

Part 01: Steganalyzer
M1: Basic CNN
No noise extraction as a pre-processing step
CNN Classification with no noise pre-processing
OR
M2: GNCNN
No noise extraction as a pre-processing step
Image pre-processing layer using HPF
+
Convolutional layers
+
Classification layers
OR
M3: Xu-Net
Noise extraction using HPF as a pre-processing step
Convolutional Modules
+
Linear classification module
OR
M4: DCNN + CNN
Denoising CNN as a noise extractor
Classification CNN based on the residual noise obtained
Steganography absent
Steganography present
Part 02: Extraction Module
Data Extraction Module
TEXT
Part 03: Text-Classifier
Embedding layer
+
LSTM
+
Classification layer
Malicious code
Benign text
Stegware Detected!

Figure 3.2: Overview of the proposed System

Part 01: CNN Steganalyzer

3.2.1 Model 01

The first model is the implementation of a simple CNN, as shown in the figure **3.3**. It consists of 3 convolutional layers, 3 max pooling layers, one flatten layer, and two fully connected layers.

The input image, an RGB image with dimensions 224x224, is passed through the first Conv2D layer, applying 32 filters of size 3x3. This generates 32 feature maps, each highlighting different learned patterns or features from the image.

The feature maps then go through a MaxPooling2D layer to reduce their spatial dimensions by a factor of two, using a 2x2 pooling window.

Next, the 32 feature maps from the previous step are fed into the second Conv2D layer, applying 64 filters of size 3x3. This produces 64 new feature maps, capable of capturing more complex and abstract features.

The process is repeated for the third Conv2D and MaxPooling2D layers, using 128 filters and resulting in 128 feature maps.

The Flatten layer converts the feature maps into a 1-dimensional vector with 86,528 units, representing the learned filters in a sequential manner.

The flattened vector is then connected to a Dense layer with 128 units, utilizing the ReLU activation function to enhance the model's ability to capture complex relationships among the filters.

Finally, the output of the previous Dense layer is connected to a second Dense layer with a single unit and the Sigmoid activation function. This unit represents the probability of the input image belonging to either the clean or steganographic class.

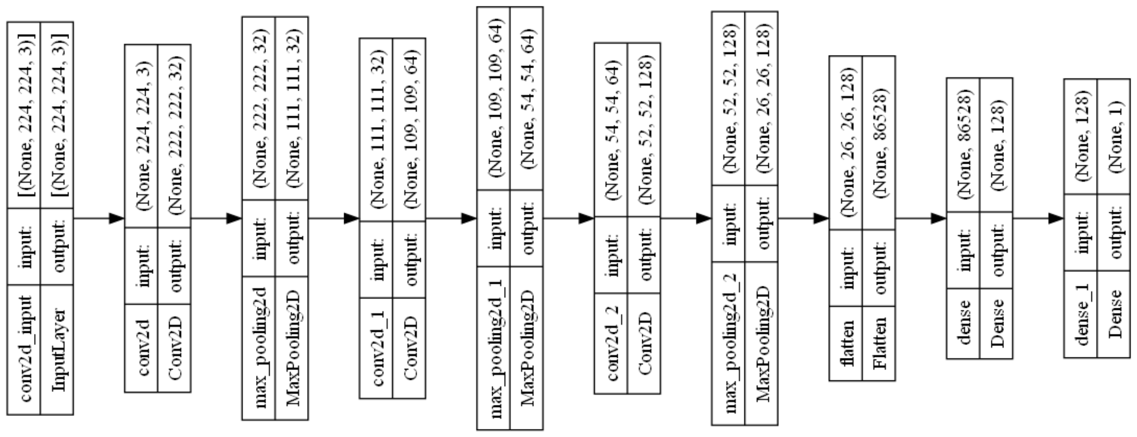

conv2d_input | InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
conv2d | Conv2D | input: (None, 224, 224, 3) | output: (None, 222, 222, 32)
max_pooling2d | MaxPooling2D | input: (None, 222, 222, 32) | output: (None, 111, 111, 32)
conv2d_1 | Conv2D | input: (None, 111, 111, 32) | output: (None, 109, 109, 64)
max_pooling2d_1 | MaxPooling2D | input: (None, 109, 109, 64) | output: (None, 54, 54, 64)
conv2d_2 | Conv2D | input: (None, 54, 54, 64) | output: (None, 52, 52, 128)
max_pooling2d_2 | MaxPooling2D | input: (None, 52, 52, 128) | output: (None, 26, 26, 128)
flatten | Flatten | input: (None, 26, 26, 128) | output: (None, 86528)
dense | Dense | input: (None, 86528) | output: (None, 128)
dense_1 | Dense | input: (None, 128) | output: (None, 1)

Figure 3.3: The structure of the basic proposed CNN

3.2.2 Model 02

The second model is the implementation of GNCC, a variation of CNN proposed by Qian et al. it is considered to be the first CNN to use Gaussian function as the activation function, from which the name was derived [39]. This model is composed of a pre-processing layer, multiple convolutional layers and a classification layer as depicted in figure **3.5**.

Image pre-processing: CNNs are yet not developed enough to extract certain statistical features, and the noise presented by steganography being a weak noise, a basic CNN no matter how deep it is, will most likely fail to capture it. For this reason, the GNCNN model starts with an image pre-processing layer. This layer applies a high-pass filter to the input image, aiming to enhance the images noise and reduce the impact of image content. we denote ***I*** as image, ***R*** as image after high-pass filtering (usually referred to as residual image), and ***Kv*** as the HPF:

$$R = kv * I$$

The high-pass filter is defined as follow:

$$kv = \frac{1}{12}\begin{pmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & -1 \end{pmatrix}$$

An HPF is a kernel matrix, based on the Laplacian of Gaussian (LoG) filter. The LoG filter is designed to enhance the high-frequency components (edges, details) in an image while suppressing the low-frequency components (smooth regions). The matrix values can be obtained using the Laplacian operator, which is a discrete approximation of the second derivative as follow:

$$L = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

where f is the image function, and $\partial^2/\partial x^2$ and $\partial^2/\partial y^2$ represent the second derivatives in the x and y directions, respectively.

We then start with a 3x3 neighbourhood for the approximation, where each pixel contributes to the computation of the Laplacian value at its central position.

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

This operator assigns a weight of 0 to the central pixel and weights of 1 to its four neighbouring pixels. The neighbouring pixels are in the horizontal and vertical directions, reflecting the second derivative approximation. The negative weight in the central position is used to subtract the average value of the neighbouring pixels from the central pixel, enhancing the edges.

To extend this 3x3 Laplacian operator to a 5x5 matrix, the same pattern can be repeated while adding additional rows and columns of zeros around it. This expansion ensures that the Laplacian operator covers a larger neighbourhood for better edge detection. The resulting 5x5 matrix is the ***Kv*** we previously presented.

Convolutional layer: After the image processing layer, convolutional layers are used to capture dependencies among local and global regions of the stego signal. Each layer performs convolution, non-linearity using the Gaussian activation function, Generally, it is chosen to be a sigmoidal function such as the logistic sigmoid or the $tanh()$ sigmoid function in traditional CNNs. But in this work, a Gaussian function is used, which can be express as:

$$f(x) = e^{-\frac{x^2}{\sigma^2}}$$

where σ is a parameter that determines the width of the curve. A neuron with this activation function will produce a significant positive response only when the input falls into a small interval around zero.

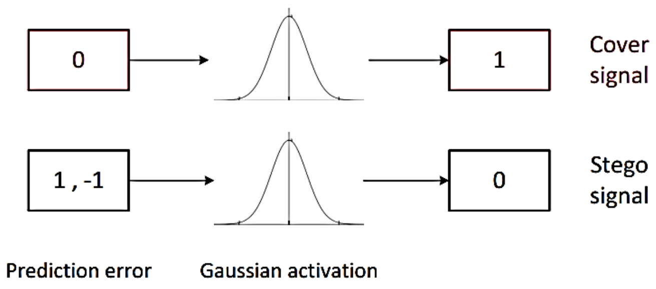


Figure 3.4: The Gaussian Function [39]

The resulting activations are then passed to the pooling part of the layer, two conventional choices for pooling are available, average pooling and max pooling. While max pooling captures the strongest activation, average pooling considers all activations in the region, making it suitable for steganalysis.

Classification layer: Fully connected layers are employed for classification. The learned features from the convolutional layers are passed to these layers, and a softmax activation function is used to produce a probability distribution over the class labels (two-way softmax in this case).

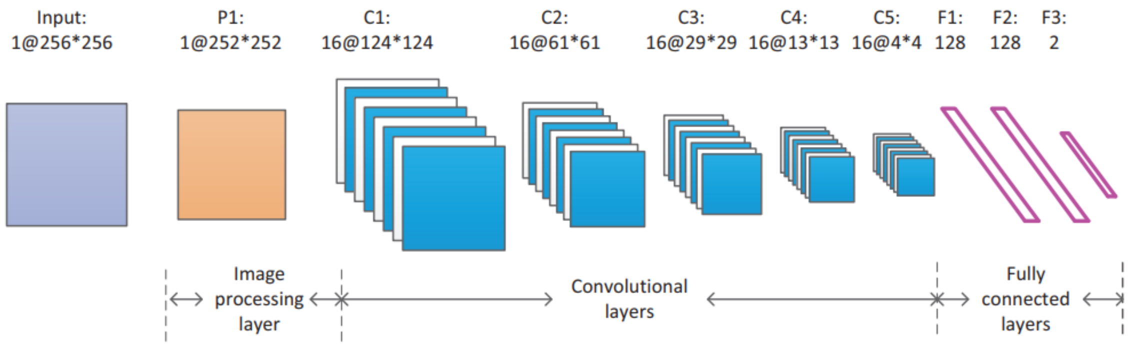


Figure 3.5: Structure of GNCNN [39]

3.2.3 Model 03

The third model is based on the work proposed by Xu et al., referred to as the "Xu-net" [40]. It's important to clarify that this model is from 2016 and not 2017. The improved architecture introduced in 2017 specifically operates on JPEG images, utilizing a DCT transformation during the pre-processing stage, as JPEG is a lossy format. The depicted model, shown in figure **3.6**, consists of a pre-processing step that involves the use of a High Pass Filter (HPF) similar to the GNCNN model. This is followed by 5 groups of layers, a global average pooling, and a linear classification module.

Pre-processing step: As mentioned earlier, an HPF is employed as a pre-processing step during the generation and loading of the images.

Convolutional modules: The CNN consists of a convolutional module responsible for transforming the images into feature vectors. This module is divided into five groups of layers, referred to as "Group 1" to "Group 5" in the accompanying figure. Each group starts with a convolutional layer that generates feature maps and ends with an average pooling layer for local averaging and subsampling (except for Group 5).

Activation Functions: To enhance the power of statistical modelling, different activation functions are employed in different groups. Group 1 and Group 2 use the hyperbolic tangent ($tanH()$) activation function, while Group 3, Group 4, and Group 5 use the rectified linear unit (ReLU) activation function. The activation functions introduce non-linearities to capture complex relationships in the data.

ABS Layer: Within Group 1, an absolute activation (ABS) layer is inserted to enforce the statistical modelling to consider the symmetry (sign) present in the noise residuals. This layer helps to capture and utilize the symmetry information during the steganalysis process.

Batch-Normalization (BN): To aid in the training process and prevent the CNN from getting stuck in poor local minima, batch-normalization is performed immediately before each non-linear activation layer. Batch-normalization normalizes the input data to have zero mean and unit variance, which helps stabilize and accelerate the training of the CNN.

Global Averaging: In Group 5, a pooling layer performs global averaging, which collapses each spatial map into a single element. This results in a feature vector of size 128.

Linear Classification Module: The linear classification module follows the convolutional module. It consists of a fully-connected layer (no hidden layers) and a softmax layer. It is responsible for making the final predictions based on the learned features.

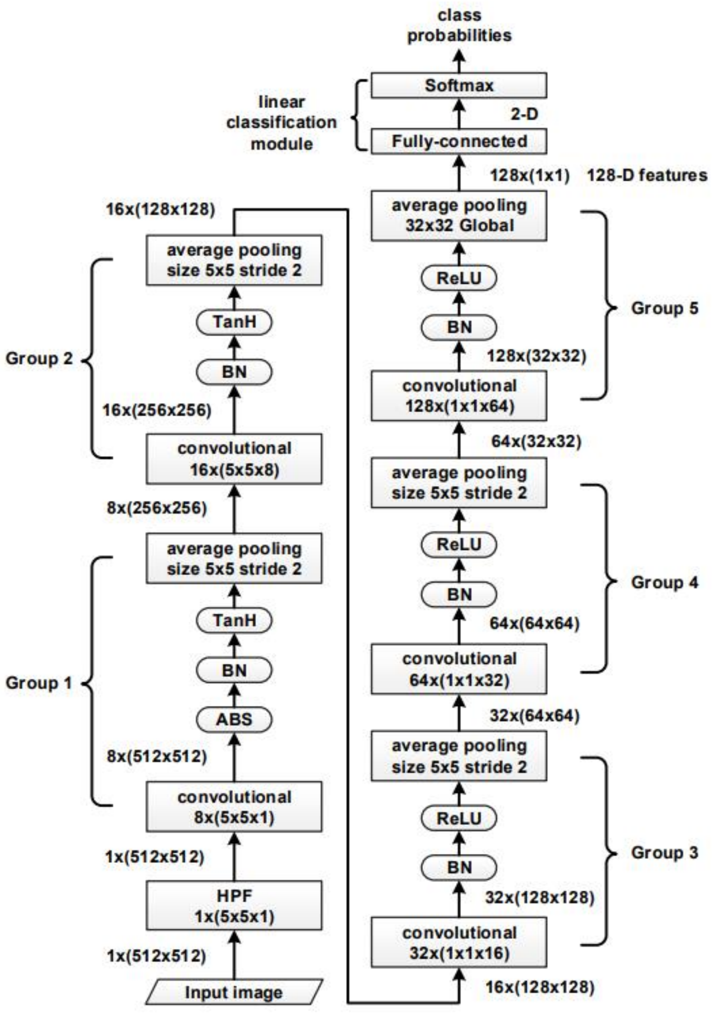


Figure 3.6: The structure of the proposed Xu-net [40]

3.2.4 Model 04

The fourth model is based on a proposed approach by Brijesh Singh on his thesis [41], he proposed the usage of an architecture composed of two modules: a denoising CNN and a classifying CNN as shown in figure **3.7**.

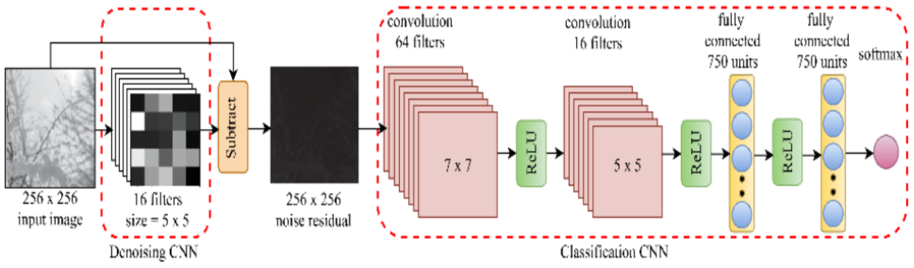


Figure 3.7: Global structure of the fourth model [41]

Denoising CNN: this denoiser comprises a single convolutional layer with 16 kernels of size 5×5. To preserve the stego noise, which is a relatively weak signal, no pooling layer is included in the network. The stride is set to 1 to ensure convolution over the entire image without padding. The DCNN is trained to predict a denoised cover image from a stego image. The architecture of the proposed DCNN is illustrated in Figure **3.8.**

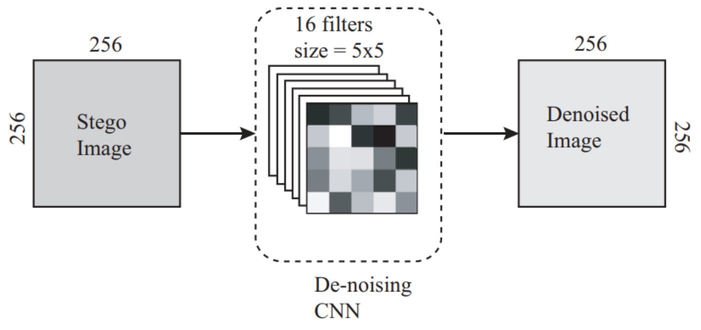


Figure 3.8: The structure of the DCNN [41]

Classifying CNN: In the second module, a shallower CNN is utilized as a steganalytic classifier, comprising two convolutional layers and two fully connected layers with varying filter sizes. The first layer takes a 256x256 noise residual as input and employs 64 filters of size 7x7, producing 64 feature maps of size 128x128. The second layer uses 16 filters of size 5x5 to extract more detailed features, resulting in 16 feature maps of size 64x64. The fully connected layers consist of 750 neurons each, and a softmax layer is employed for binary classification. ReLU activation and dropout with a probability of 0.8 are applied to mitigate overfitting, while pooling layers are removed to preserve the weak stego noise.

Note:

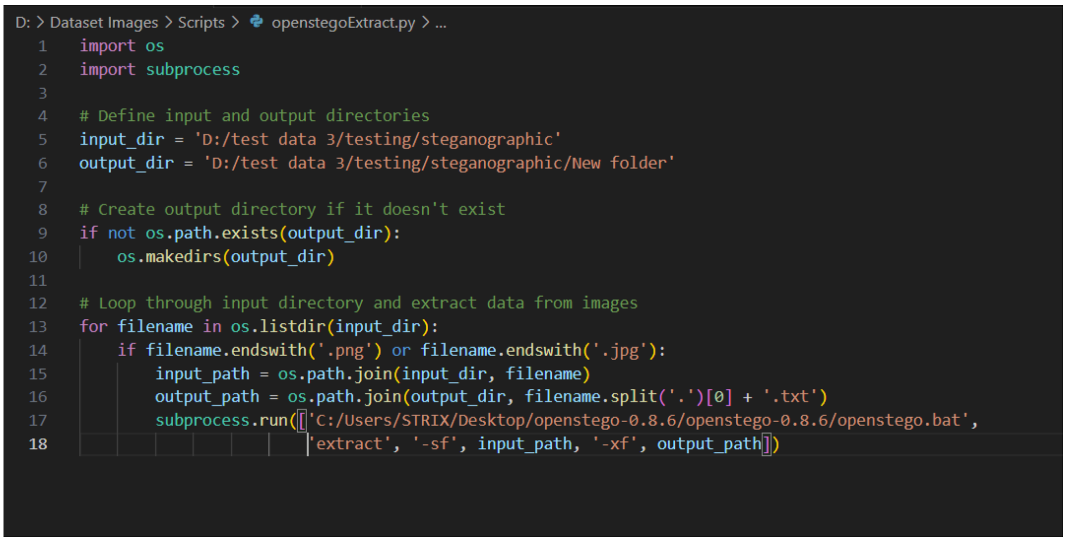
We would like to emphasize that we implemented two variations of the proposed approach:

1. Denoising CNN combined with a CNN classifier.
2. Autoencoder integrated with a CNN classifier.

The motivation behind this decision was driven by the widespread use of autoencoders for denoising tasks, and our aim was to investigate whether autoencoders could effectively capture the subtle noise introduced by the steganography embedding process.

Part 02: Extraction Module

For the second part of our system, we employed a selected extraction tool to retrieve the data from the images identified as steganographic by the initial steganalyzer. In consideration of the primary focus of this work, which is not cryptography, we opted to use OpenStego. This tool was also utilized for encrypting our data and embedding it within the cover images.


D: > Dataset Images > Scripts > openstegoExtract.py > ...

```python
import os
import subprocess

# Define input and output directories
input_dir = 'D:/test data 3/testing/steganographic'
output_dir = 'D:/test data 3/testing/steganographic/New folder'

# Create output directory if it doesn't exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Loop through input directory and extract data from images
for filename in os.listdir(input_dir):
    if filename.endswith('.png') or filename.endswith('.jpg'):
        input_path = os.path.join(input_dir, filename)
        output_path = os.path.join(output_dir, filename.split('.')[0] + '.txt')
        subprocess.run(['C:/Users/STRIX/Desktop/openstego-0.8.6/openstego-0.8.6/openstego.bat',
                        'extract', '-sf', input_path, '-xf', output_path])
```

Figure 3.9: Extraction script snippet in Python

The code presented in the figure **3.9** demonstrates an example of utilizing the OpenStego extractor to retrieve data from a batch of steganographic images and save it to a new folder. It is important to clarify that OpenStego uses the AES encryption algorithm, which is widely recognized as one of the most secure encryption algorithms available. Any attempts to circumvent this encryption and extract the data would require an entirely separate work.

Part 03: Text Classifier

For the classification task of the text files, and due to the contrast difference between the plain text files and the code files, we opted for a fairly simple RNN illustrated in figure **3.10**. This RNN is composed of an embedding layer follower by an LSTM layer and finally a dense layer.

Embedding layer: this layer is responsible for mapping the input sequence of words (represented as integers) to dense vectors of fixed size. It learns the embeddings or representations of words in a continuous space, allowing the model to capture the semantic relationships between words.

LSTM (Long Short-Term Memory): this layer is a type of recurrent layer that can capture long-term dependencies in sequential data. It has memory cells that can retain information over longer sequences, enabling the model to learn patterns and dependencies in the input text.

Dense layer: this layer with a sigmoid activation function produces a single-unit output with a range between 0 and 1. It will classify the text files as either malicious code or plain harmless text.

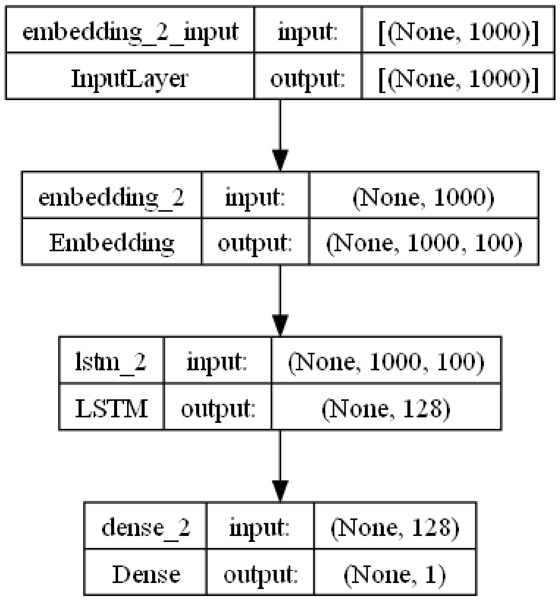


Figure 3.10: Structure of our proposed text-based classifier

3.3 Conclusion

This chapter was dedicated to the exploration of various proposed models, their detailed structures, and the necessary modifications made to construct the modules that compose our system. In the subsequent chapter, Chapter Four, we will delve into the datasets employed, as well as the tests conducted and the results obtained from implementing these models on both PNG images and text files.

Chapter 04

Test and Validation

4.1 Introduction

In the following chapter, we will discuss the steps we took to conduct our research, beginning with the construction of our dataset and progressing to the training, testing, and comparison of the results obtained from various implementations of the architectures discussed in the previous chapter.

4.2 Tools and working environment

4.2.1 Hardware specifications

to train our models, two different laptops were used, their specifications are the following:

Laptop 01:

– OS: Windows 10

– CPU: i7-7700HQ CPU @ 2.80 GHz

– GPU: NVIDIA GTX 1050 @ 4.0 GB

– RAM: 16.0 GB

Laptop 02:

– OS: Windows 11

– CPU: i7-8750H CPU @ 2.20 GHz

– GPU: NVIDIA GTX 1070 @ 8.0 GB

– RAM: 16.0 GB

4.2.2 Libraries and software

Tool/Library	Description
Python	Chosen high-level programming language.
OpenStego	Steganography tool for concealing data in digital images.

TensorFlow	Library for ML and neural networks.
Keras	High-level neural networks API.
NumPy	Library for numerical computations.
Matplotlib	Plotting library for creating visualizations.
OpenCV	Library for computer vision and image processing.
SciPy	Library for scientific and technical computing.
Scikit-learn	Library for ML algorithms.
Markovify	Library for generating Markov chain-based test.

Table 4.1: Libraries and tools used.

4.3 Dataset

4.3.1 Steganalyzer Dataset

Preparing the appropriate dataset is a crucial step in building and training any deep learning model. In our case, several specifications had to be taken into consideration to better represent real-life scenarios:

- Ensuring an equal distribution between clean images and steganographic images (referred to as "stego" images from now on).
- Considering the frequency of usage in terms of file formats. According to statistical studies conducted by w3techs.com, the PNG image format has consistently topped the charts for the past 5 years, as shown in the Figure **4.1**.
- Using the RGB colour model for images.

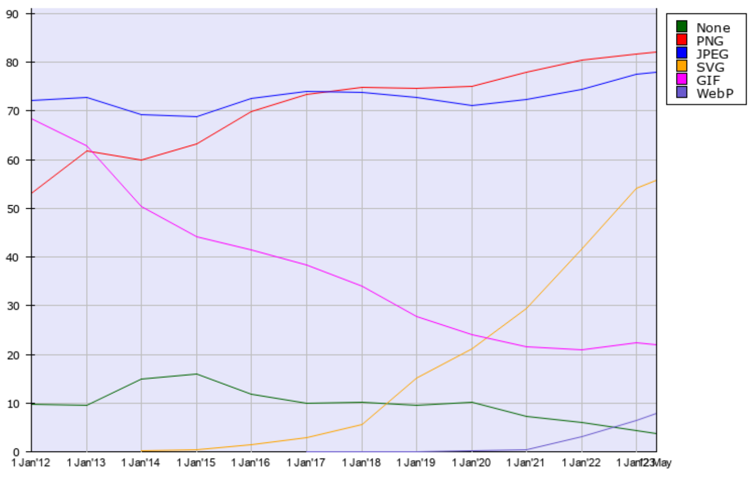


Figure 4.1: Usage of image formats for websites [42]

BOSSbase, despite being widely used in steganalysis, did not meet our requirements

4.3.1.1 BOSSbase Dataset

BOSSbase is a training database consisting of 9074 grayscale images in the PGM format with a resolution of 512x512 pixels [43]. It was initially released on June 28th, 2010, as part of the challenge with the same name, "Break Our Steganographic System". Please refer to the website and the accompanying paper for more details. There are several reasons why we chose not to use this dataset:

- The images were captured by only 7 cameras in similar circumstances, which does not provide the desired diversity and representativeness for our dataset.
- The images in the dataset are grayscale, whereas we require RGB images.
- The format of the dataset's images did not present a significant issue, as we could have solved it by converting the images from PGM to PNG.

4.3.1.2 Our diverse dataset

For the reasons previously mentioned, we made the decision to create our own dataset from scratch. To achieve this, we undertook a series of meticulous steps shown in figure **4.2**.

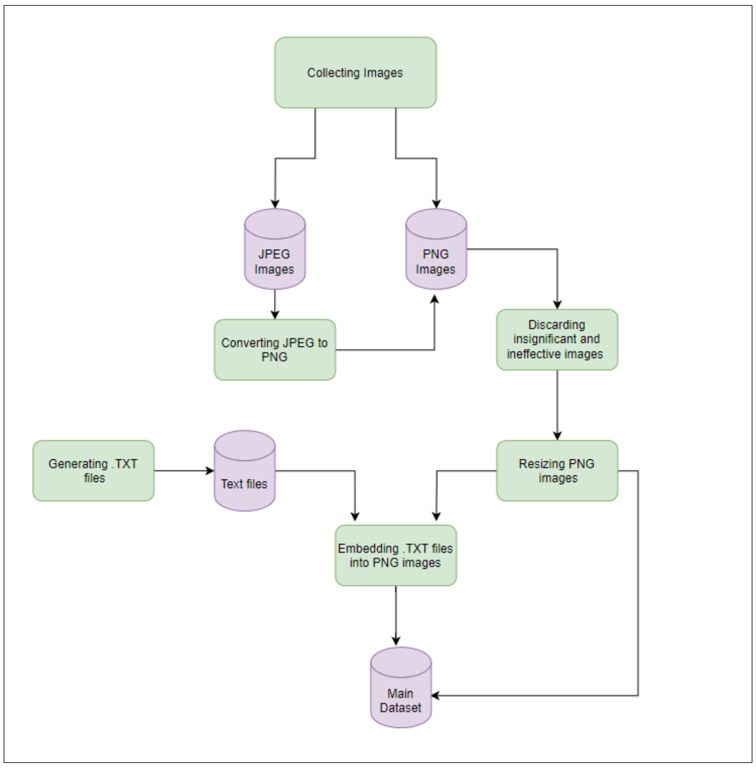


Figure 4.2: A scheme representing the steps of data collection

Algorithm 01: Preparing Dataset

```
 INPUT: Data ← PNG Images = N
 OUTPUT: Clean images, Steganographic images
1.  Ω = ∅ initialize pool of PNG images to empty
2.  φ = {Text files}
3.  For i images in N do
4.      If format of image i <> 'PNG' then
5.          Convert image i to PNG
6.          Ω += i add the resized image to the pool
7.      Else
8.          Ω += i
9.      End if
10. End for
11. For i image in Ω do
12.     If Size of i < 400KB then
13.         Ω -= i remove image i from pool
14.     Else
15.         Resize image i
16.     End if
17. Ω = Ω / 2    Devise the pool of images by 2 to ensure an equal distribution
    between clean and steganographic images in our dataset.
18. For i image in Ω do
19.     For j text file in φ
20.         Embed j text file into i image
21.     End for
22. End for
```

Table 4.2: Data Collection Algorithm

Algorithm **4.2** illustrates the detailed steps of constructing our dataset, and the table below presents three of its variants, with the main dataset being the third one. The first and second datasets can be considered subsets of the third dataset, which will be discussed in detail when we get to the results of the experiments. We selected an embedding rate of 0.5 for the main dataset, while for the sub-datasets, we experimented with both 0.5 and 0.8 embedding.

	Dataset 01		**Dataset 02**		**Dataset 03**
Embedding rate	0.5	0.8	0.5	0.8	0.5
Training	1442		7000		21000
	70%		70%		70%
Testing	308		1500		4500
	15%		15%		15%
Validation	308		1500		4500
	15%		15%		15%
Total	2058		10000		30000

Table 4.3: Table of embedding rates and sub-datasets

Evaluating our proposed dataset:

Since our proposed dataset was built from scratch, and in order to ensure the successful embedding process and verify the impact of the LSB embedding process on the composition of steganographic images, we used various statistical metrics to evaluate it, these metrics are the following:

Chi-square

The chi-square test measures the difference between the observed and expected pixel frequencies in an image. It evaluates the hypothesis that the distribution of pixel values in the clean and stego images is significantly different. The following clean image (Figure **4.3**) has been randomly selected from our dataset, along with its corresponding stego image (Figure **4.4**). when observing the Chi-square diagram (Figure **4.5**), we can see a subtle difference between the two.

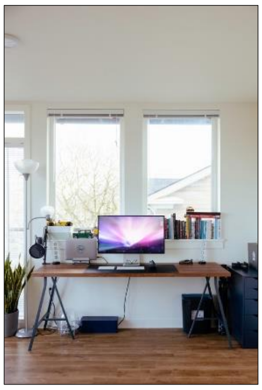


Figure 4.3: A clean image from our dataset

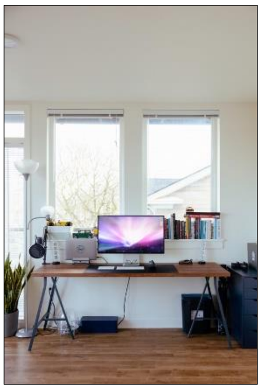


Figure 4.4: A stego image from our dataset

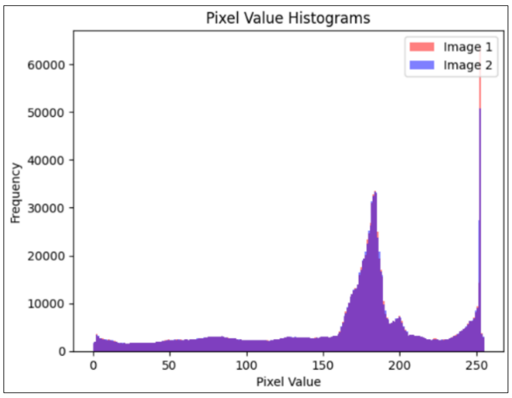


Figure 4.5: The Chi-square Diagram

Histogram

Histogram analysis is another valuable statistical metric for comparing clean and stego images. A histogram represents the frequency distribution of pixel values in an image. By examining the histograms of clean and stego images, you can identify differences in their pixel value distributions. The following (Figure **4.6**) is a histogram of the same previous images overlapped and the difference between the two.

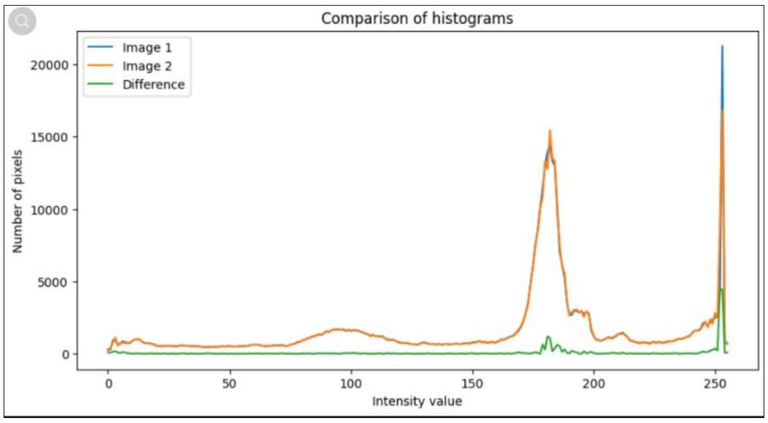


Figure 4.6: The histogram of pixel values

MSE, PSNR and SSIM

MSE: MSE stands for Mean Squared Error. It is a commonly used metric in image processing and other fields to measure the average squared difference between the pixel values of two images or sets of data. In the context of comparing images, the MSE quantifies the dissimilarity or distortion between a reference image (e.g., clean image) and a test image (e.g., stego image). It calculates the average of the squared differences between corresponding pixel values in the two images.

PSNR: PSNR stands for Peak Signal-to-Noise Ratio. It is a widely used metric for measuring the quality or fidelity of reconstructed or compressed images. PSNR provides a quantitative assessment of the difference between a reference image and a distorted or reconstructed image by considering both the error magnitude and the dynamic range of pixel values.

SSIM: SSIM stands for Structural Similarity Index. It is a widely used metric for measuring the perceived similarity or quality of images, taking into account both structural information and pixel-wise differences. SSIM aims to mimic human perception by considering the relationships between neighbouring pixels and the overall structure of the image.

When comparing the previous pictures, we obtained the following results:

Metric	**Stego vs clean images**	**Identical images**
MSE	0.99	0.0
PSNR	52.06	361.20
SSIM	0.99	1.0

Table 4.4: Comparison of MSE, PSNR and SSIM

Note on how to read the results:

- A lower MSE value indicates a smaller difference between the pixel values of the two images and suggests a higher similarity or less distortion.
- A higher PSNR value indicates a smaller difference or distortion between the images and suggests better quality or higher fidelity. Conversely, a

lower PSNR value indicates more noticeable differences or artifacts in the reconstructed or distorted image compared to the reference image.

– For SSIM:

SSIM = 1	SSIM > 0.9	SSIM > 0.8	SSIM > 0.7	SSIM < 0
Perfect similarity	Very high level of similarity	Reasonably good level of similarity	Moderate level of similarity	The images are dissimilar

Table 4.5: SSIM values and their readings

After evaluating our proposed dataset using the metrics mentioned above, we have ensured that any variations in the results cannot be attributed to flaws in the construction of our steganalyzer dataset.

4.3.2 Text-Classifier Dataset

Constructing a dataset from scratch for our text classifier presented several challenges due to the following reasons:

4.3.2.1 Source code-based malware analysis

Seeing that the problem at hand requires a specific type of malware detection and classification that is not widely used, static malware analysis becomes a relevant approach. Static malware analysis, as mentioned in the first chapter, encompasses the following techniques:

- **Code and behaviour analysis:** This involves examining the code to identify patterns, signatures, or known malicious behaviour. It includes API calls, function calls, system calls, and detecting obfuscated code.
- **Structural analysis:** This means studying the structure of the malware code, such as the presence of packers, obfuscation techniques, or anti-analysis mechanisms. It helps in comprehending how the malware attempts to evade detection or analysis.

4.3.2.2 Our dataset

Unfortunately, the availability of malware source code datasets is limited, which led us to create our own dataset through web scraping of malware source code repositories. However, this approach proved to be a time-consuming and yielded fewer promising results. The scarcity of publicly available malware source codes poses legal and ethical concerns, as they are rarely disclosed openly. To overcome this limitation, we used augmentation techniques that involved random modifications, such as variable name changes, code block rearrangements, and the insertion of random lines of commented code.

	Malicious code	**Plain Text**
Training	175	175
Validation	820	820
Testing	175	175

Table 4.6: Our dataset for the text classifier

4.4 Results and Discussion

In this section of the chapter, we present the results obtained after training, validating, and testing all of the previously mentioned models on the dataset we detailed. We also discuss potential reasons and justifications for the performance results of each model.

Results

4.4.1 Part one: Steganalyzer

We begin with our steganalyzer. The following tables present the accuracy achieved for each of the four implemented models. Each model trained on the main dataset and its variant sub-dataset and with corresponding bits per pixel (bpp):

4.4.1.1 Model 01: Basic CNN

As depicted in the table **4.7** below, the first CNN results demonstrate random guessing, as indicated by the confusion matrix occasionally classifying all instances as either clean or steganographic:

	Dataset 01		**Dataset 02**		**Main Dataset**
bpp	0.5	0.8	0.5	0.8	0.8
Accuracy	50%	52%	50%	50%	50%

Table 4.7: Results for the CNN first model.

4.4.1.2 Model 02: GNCNN

Despite incorporating a pre-processing layer utilizing high-pass filtering (HPF) to highlight the stego noise, GNCNN did not outperform a basic CNN without the noise extraction step or pre-processing. Although it exhibited promising results in tests conducted on JPEG images, the performance of GNCNN did not show any improvements in the case of PNG images. The results are presented in Table **4.8**:

	Dataset 01		**Dataset 02**		**Main Dataset**
bpp	0.5	0.8	0.5	0.8	0.8
Accuracy	50%	54%	50%	50%	50%

Table 4.8: Results for the second model GNCNN

4.4.1.3 Model 03: Xu-Net

Xu-net, on the other hand, demonstrated slightly improved results compared to the first two models. As shown in Table **4.9**, the results indicate performance levels slightly higher than random guessing

	Dataset 01		**Dataset 02**		**Main Dataset**
bpp	0.5	0.8	0.5	0.8	0.8
Accuracy	52%	61%	54%	54%	53%

Table 4.9: Results for the third model, Xu-Net.

4.4.1.4 Model 04: DCNN + Classifier

The fourth model showcased the most promising results for the PNG dataset, achieving an impressive accuracy of 75%. It clearly outperformed the other models, as backed-up by the data presented in both table **4.10 and** table **4.11**.

We'd also like to note that we considered the usage of autoencoders to capture the noise instead of the proposed denoising CNN, however, the results yielded were disappointing to say the least with a loss of **0.087**.

DCNN

We note that for this model, we relied solely on the main dataset, training the denoiser we got:

MAE	0.0409
Loss	0.0056

Table 4.10: MAE and loss results for the denoiser

Classifier

As for the Classifier we obtained promising results, indicating that this classifier did better at classifying the residual noise than the other two models:

Accuracy	75.09%

Table 4.11: Accuracy achieved for the fourth model classifier

4.4.2 Part two: Text Classifier

The contrast between plain text and code is substantial and distinct, allowing our simple text classifier to achieve an exceptionally high accuracy of 99.70% as shown in table **4.12**:

Accuracy	99.70%

Table 4.12: Accuracy for text-based classifier.

Discussion

In this section, we provide analysis and discussion of the results presented for each part and model. Our objective is to identify potential factors that may account for the varying performances and outcomes, whether they be poor or promising.

Model 01:

Model 01 served as the initial foundation for this work, where we assessed the performance of a relatively simple yet moderately deep CNN. The aim was to evaluate its ability to detect and learn the features and distortions introduced during the embedding process. However, the results showed that the model achieved an accuracy equivalent to random guessing. Notably, in certain datasets, it exhibited a bias towards the 'clean' class, while in others, it leaned towards the 'steganographic' class.

Model 02:

GNCNN, which incorporated the consideration of noise present in both stego and clean images, demonstrated potential in theory, as we previously discussed. However, when applied to a dataset of PNG images, it did not yield satisfactory results. One possible explanation for this outcome is that the classifier was unable to effectively learn and distinguish the subtle noise patterns that differentiate the clean noise from the stego noise, despite the application of a high-pass filter (HPF).

Model 03:

Xu-net's results provided evidence to support our hypothesis that the inability of GNCNN to perform well was primarily attributed to the classifier component of the architecture, rather than the HPF. It is worth noting that both GNCNN and Xu-net incorporated the use of HPF as a pre-processing step, either as a customized layer in the case of GNCNN or a function applied during data loading. This observation suggests that the difference in performance between the two models can be attributed to Xu-net's classifier part, which demonstrated a better capability to learn and identify the distinctive stego noise patterns to some degree.

Model 04:

The fourth model, which yielded the best results, owes its success to two key factors. Firstly, the denoising module played a crucial role in effectively capturing the subtle stego noise present in both clean and stego images. By leveraging the ground truth of clean images and starting from a set of stego images, the denoising module generated denoised images with remarkable similarity to the original covers. Secondly, the classifier component of this model was slightly deeper compared to the previous models. This deeper architecture likely contributed to its enhanced ability to learn and classify the distinctive features associated with stego and clean images, further boosting its overall performance.

Autoencoder:

While one may argue that a loss of 0.087 is acceptable for a denoising autoencoder, it is important to note that the training process was not effective. The training loss and validation loss exhibited minimal changes over a period of 10 epochs. Additionally, when the autoencoder was tested on images, the results were found to be unsatisfactory. A plausible explanation for this could be that the training of autoencoders is insufficient to effectively capture and denoise the subtle noise introduced during the embedding process of steganography techniques. The complexity of the steganographic embedding may require further training or more sophisticated approaches to achieve optimal denoising performance.

4.5 Conclusion

This chapter concluded the entire work with promising results that have not been achieved before in the field of steganalysis of PNG images using deep learning. However, it is important to note that this achievement is only a part of the overall goal of this work. The trained models, along with the detailed implementation, the appropriate data extractor, and the accurate method of classifying the extracted data, contribute to the development of a robust and high-performing system to combat the emerging threat of stegomalware in images

General Conclusion

We embarked on this work with the intention of developing an approach to detect the evolving trend in malware techniques that aim to remain hidden in plain sight, commonly known as stegware. Our process involved conducting thorough bibliographic research, collecting and constructing our own dataset, and training five different models: four for the steganalyzer component and one for the text classifier component. The main goal was to create a system capable of dissecting stegware into its individual components and addressing each component using the most suitable methods. Our work concluded with results that were deemed reasonably acceptable, serving as a foundation for future enhancements and advancements in this field.

In this work, we also ventured into the realm of PNG images, a widely utilized format by internet users and certain stegware variants. Despite the limited research conducted on this format, we aimed to demonstrate the feasibility of applying steganalysis to PNG images and showcase the potential for achieving significant results. Our results highlight the importance of further exploration and investigation in this area.

In conclusion, we would like to draw attention to the fact that the sub-field of stegware detection and analysis remains largely unexplored. This raises significant concerns considering the increasing adoption of steganography techniques by malware families. The potential for any malware to easily incorporate this technique and evade detection poses a serious threat. It is imperative that further research and development efforts be dedicated to advancing stegware detection methods to stay ahead of these evolving threats and ensure the security of digital systems and networks.

References

[1]. Judge, J C. *Steganography: Past, present, future*, 2001, https://doi:10.2172/15006450

[2]. White William. *The microdot: History and Application*. Phillips Publications, 1992.

[3]. Cole, Eric. “Covert Communication: It’s All Around You.” *Hiding in plain sight: Steganography and the art of covert communication*, Wiley Pub., New York 2003, pp. 1–7.

[4]. Shih, Frank Y., and Venkata Gopal Edupuganti. “Differential Evolution Based Algorithm for Breaking the Visual Steganalytic System.” *Multimedia Security: Watermarking, Steganography, and Forensics*, CRC Press, Taylor &amp; Francis Group, Boca Raton, 2013, pp. 245–255.

[5]. Cheddad, Abbas, et al. “Digital Image Steganography: Survey and analysis of current methods.” *Signal Processing*, vol. 90, no. 3, 2010, pp. 727–752, https://doi.org/10.1016/j.sigpro.2009.08.010

[6]. Majeed, Mohammed Abdul, et al. “A Review on Text Steganography Techniques.” *Mathematics*, vol. 9, no. 21, 2021, p. 2829, https://doi.org/10.3390/math9212829

[7]. Cox, Ingemar J. *Digital Watermarking and Steganography*. Morgan Kaufmann Publishers, 2008.

[8]. Piper, Frederick Charles, and Sean Murphy. *Cryptography: A Very Short Introduction*. Oxford University Press, 2002.

[9]. AL-Ani, Zaidoon Kh., et al. "Overview: Main Fundamentals for Steganography." *ArXiv*, 2010, abs/1003.4086.

[10]. Amitava Podder, et al. “Steganography Techniques - an Overview.” *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2022, pp. 323–327, https://doi.org/10.32628/cseit228642

[11]. Juneja, Mamta, and Parvinder S. Sandhu. “An Improved LSB Based Steganography Technique for RGB Color Images.” *International Journal of Computer and Communication Engineering*, 2013, pp. 513–517, https://doi.org/10.7763/ijcce.2013.v2.238

[12]. Prasad, M. Sitaram, et al. "A novel information hiding technique for security by using image steganography." *Journal of Theoretical and Applied Information Technology*, vol. 8, no. 1, 2009, pp. 35-39. Accessed 10 May 2023. Available at: www.jatit.org/volumes/research-papers/Vol8No1/6Vol8No1.pdf

[13]. Amitava Podder, et al. “Steganography Techniques - an Overview.” *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2022, pp. 323–327, https://doi.org/10.32628/cseit228642

[14]. Shih, Frank Y., and Mayra Bachrach. “Survey of Image Steganography and Steganalysis.” *Multimedia Security: Watermarking, Steganography, and Forensics*, CRC Press, Taylor &amp; Francis Group, Boca Raton, 2013, pp. 201–214.

[15]. Wu, Da-Chun, and Wen-Hsiang Tsai. “A Steganographic Method for Images by Pixel-Value Differencing.” *Pattern Recognition Letters*, vol. 24, no. 9–10, 2003, pp. 1613–1626, https://doi.org/10.1016/s0167-8655(02)00402-6

[16]. Stuti Goel, Stuti Goel. “A Review of Comparison Techniques of Image Steganography.” *IOSR Journal of Electrical and Electronics Engineering, vol*. 6, no. 1, 2013, pp. 41–48, https://doi.org/10.9790/1676-0614148

[17]. Gupta, D., and Siddhartha Choubey. "Discrete Wavelet Transform for Image Processing." *International Journal of Emerging Technology and Advanced Engineering*, vol. 4, 2015, pp. 598-602.

[18]. Karampidis, Konstantinos, et al. “A Review of Image Steganalysis Techniques for Digital Forensics.” *Journal of Information Security and Applications*, vol. 40, 2018, pp. 217–235, https://doi.org/10.1016/j.jisa.2018.04.005

[19]. Nissar, Arooj, and A.H. Mir. “Classification of Steganalysis Techniques: A Study.” *Digital Signal Processing*, vol. 20, no. 6, 2010, pp. 1758–1770, https://doi.org/10.1016/j.dsp.2010.02.003

[20]. Wallarm. “What Is Malware? Types and Examples.” *RSS*, 12 Apr. 2023, www.wallarm.com/what/malware-types-and-detection

[21]. Shehab, Doaa A., and Mohmmed J. Alhaddad. “Comprehensive Survey of Multimedia Steganalysis: Techniques, Evaluations, and Trends in Future Research.” *Symmetry*, vol. 14, no. 1, 2022, p. 117, https://doi.org/10.3390/sym14010117

[22]. Chaganti, Raj, et al. *Stegomalware: A Systematic Survey of Malware Hiding and Detection in Images, Machine Learning Models and Research Challenges*, 2021, https://doi.org/10.36227/techrxiv.16755457

[23]. Vasudevan, Amit, and Ramesh Yerraballi. "Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation." *Proceedings of the 29th Australasian Computer Science Conference*, 2006, pp. 311-320.

[24]. Ye, Yanfang, et al. “A Survey on Malware Detection Using Data Mining Techniques.” *ACM Computing Surveys*, vol. 50, no. 3, 2017, pp. 1–40, https://doi.org/10.1145/3073559

[25]. Ijaz, Muhammad, et al. “Static and Dynamic Malware Analysis Using Machine Learning.” *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2019, https://doi.org/10.1109/ibcast.2019.8667136

[26]. Gazet, Alexandre. “Comparative Analysis of Various Ransomware Virii.” *Journal in Computer Virology,* vol. 6, no. 1, 2008, pp. 77–90, https://doi.org/10.1007/s11416-008-0092-2

[27]. Vamshi, Ashwin. “Anatomy of a Ransomware Attack: Cerber Uses Steganography to ‘Hide in Plain Sight.’” Netskope, 10 April 2023, www.netskope.com/blog/anatomy-ransomware-attack-cerber-uses-steganography-hide-plain-sight.

[28]. Ghanshyam More, Principal Research Engineer. “Catching the Rat Called Agent Tesla.” Qualys Security Blog, 23 Dec. 2022, blog.qualys.com/vulnerabilities-threat-research/2022/02/02/catching-the-rat-called-agent-tesla

[29]. “Research, News, and Perspectives.” Trend Micro, https://www.trendmicro.com/en_us/research/17/f/adgholas-malvertising-campaign-employs-astrum-exploit-kit.html. Accessed 27 April 2023.

[30]. Haykin, Simon. *Neural Networks and Learning Machines*. Pearson India Education Services Pvt. Ltd, 2021.

[31]. GalaxyInferno. “Explaining the Components of a Neural Network [Ai].” Galaxy Inferno, 13 July 2021, https://galaxyinferno.com/explaining-the-components-of-a-neural-network-ai/

[32]. Tan, Xiao. “Libor Prediction Using Genetic Algorithm and Genetic Algorithm Integrated with Recurrent Neural Network.” *2019 Global Conference for Advancement in Technology (GCAT),* 2019, https://doi.org/10.1109/gcat47503.2019.8978299

[33]. Gu, Jiuxiang, et al. “Recent Advances in Convolutional Neural Networks.” *Pattern Recognition,* vol. 77, 2018, pp. 354–377, https://doi.org/10.1016/j.patcog.2017.10.013

[34]. Dwivedi, Rohit. “5 Common Architectures in Convolution Neural Networks (CNN).” *Analytics Steps,* www.analyticssteps.com/blogs/common-architectures-convolution-neural-networks Accessed 20 May 2023.

[35]. Trần, Lộc. *Enhancing Neural Network Explainability with Variational Autoencoders,* 2020, https://doi.org/10.2514/6.2021-1886.vid

[36]. Birla, Deepak. “Autoencoders.” *Medium,* 12 Mar. 2019, medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f

[37]. Boroumand, Mehdi, et al. “Deep Residual Network for Steganalysis of Digital Images.” *IEEE Transactions on Information Forensics and Security,* vol. 14, no. 5, 2019, pp. 1181–1193, https://doi.org/10.1109/tifs.2018.2871749

[38]. Singh, Brijesh, et al. “Steganalysis of Digital Images Using Deep Fractal Network.” *IEEE Transactions on Computational Social Systems,* vol. 8, no. 3, 2021, pp. 599–606, https://doi.org/10.1109/tcss.2021.3052520

[39]. Qian, Yinlong, et al. “Deep Learning for Steganalysis via Convolutional Neural Networks.” *SPIE Proceedings*, 2015, https://doi.org/10.1117/12.2083479

[40]. Xu, Guanshuo, et al. “Structural Design of Convolutional Neural Networks for Steganalysis.” *IEEE Signal Processing Letters*, vol. 23, no. 5, 2016, pp. 708–712, https://doi.org/10.1109/lsp.2016.2548421

[41]. Singh, Brijesh. "Spatial-Domain Image Steganalysis using Deep Learning Techniques." Thesis, Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, 2021.

[42]. “Historical Yearly Trends in the Usage Statistics of Image File Formats for Websites.”*W3Techs*,https://w3techs.com/technologies/history_overview/image_format /all/y Accessed 10 May 2023.

[43]. Bas, Patrick, et al. “break Our Steganographic System”: The Ins and Outs of Organizing Boss.” *Information Hiding*, 2011, pp. 59–70, https://doi.org/10.1007/978-3-642-24178-9_5