GAN-based Coverless Multi-Image Steganography with RGB Secret Images

K.D.M Lakshan
2024

GAN-based Coverless Multi-Image Steganography with RGB Secret Images

K.D.M Lakshan
Index No: 19000774

Supervisor : Dr. P.V.K.G Gunawardana
Co-Supervisor : Mr. T.N.B. Wijethilake

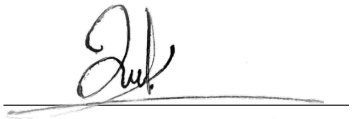
May 2024

Submitted in partial fulfillment of the requirements of the
B.Sc in Computer Science Final Year Project (SCS4224)

Declaration

I certify that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

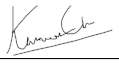
Candidate Name: K.D.M Lakshan



Signature of Candidate: 9/24/2024 Date:

This is to certify that this dissertation is based on the work of Mr. K.D.M Lakshan under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor's Name: Dr P.V.K.G Gunawardana



Signature of Supervisor: 2024/09/29 Date:

Abstract

Steganography emerges as a powerful method for secure data transfer in the digital age when information security is becoming increasingly important. Advances in this sector have opened the way for image steganography—a technique that involves embedding complete images within another, in a manner imperceptible to unsuspecting observers, assuring the communication's invisibility.

Despite breakthroughs, existing multi-image steganography models have limits that require the creation of more advanced solutions. These solutions must not only improve image quality but also improve security measures. Comprehensive research into cutting-edge techniques has revealed that Generative Adversarial Networks (GANs) significantly improve the performance of image steganography systems by increasing their embedding capacity, improving recovery accuracy, and fortifying their security protocols. As a result, the primary goal of this research is to investigate the potential of GAN-based image steganography models for advancing the multi-image steganography area, especially using a coverless technique with RGB secrete images.

The study concludes by proposing a novel coverless multi-image steganography model that leverages GANs to seamlessly embed and extract multiple secret images with minimal loss. The model's performance was rigorously evaluated using industry-standard metrics, focusing on image reproduction accuracy as measured by RMSE, MSE, PSNR, and SSIM, which were 22.50, 546.25, 21.39, and 0.66, respectively. Additionally, the model demonstrated a hiding capacity of 48 bits per pixel. Compared to traditional image steganography methods, the proposed approach not only achieves notable improvements in concealment and accuracy but also significantly enhances embedding capacity. This study advances the field of image steganography by exploring the potential of coverless multi-image steganography through the use of GANs.

Preface

This document has been produced for the partial fulfilment of the requirements of the B.Sc. in Computer Science (Hons) Final Year Project in Computer Science (SCS4124).

This dissertation introduces a novel technique that is built on Generative Adversarial Networks to address the challenges of coverless multi-image steganography with RGB secret images.

The dissertation is organized as follows: Chapter 1 presents the introduction to and background of the study, alongside a concise overview of the entire dissertation. Chapter 2 expands on this foundation, engaging in a detailed exploration of the techniques and related work within the field. Chapter 3 outlines the design of the proposed approach, while Chapter 4 details the complete implementation. A comprehensive evaluation of the proposed approach is presented in Chapter 5. Chapter 6, which provides the conclusion and future work of the research.

This dissertation represents the original work that I, along with my supervisor and co-supervisor, have conducted and hereby claim everything else mentioned in this dissertation without a specific reference to any third-party work as our own.

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Dr P.V.K.G Gunawardana, Senior Lecturer at the University of Colombo School of Computing and my co-supervisor, Mr T.N.B. Wijethilake, Lecturer at the University of Colombo School of Computing for their expertise, enthusiasm, patience, motivation and encouragement. Their guidance helped me in every aspect of my undergraduate research study. Without their guidance and support, I would not have been able to complete my work and dissertation.

I also extend my sincere thanks to the entire staff and my colleagues at the University of Colombo School of Computing (UCSC) for their constructive input, support, and collaboration, which have significantly contributed to the success of this research.

This dissertation is also dedicated to my loving family who has been an immense support to me throughout this journey of life. Without them, my effort would have been worth nothing. Their love, support, and patience inspire me to overcome all the obstacles in life and achieve goals with success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background to the Research

Steganography is an information hiding technique that has gained significant attention recently due to its ability to conceal secret information within seemingly harmless media such as images, audio, video, or text files. Unlike cryptography which involves transforming the information to make it unreadable, steganography seeks to hide the existence of the information itself. The goal of steganography is to ensure that the hidden message is not only concealed from unintended recipients but also that its existence is not detectable to anyone else.

There are various methods of steganography, including cover modification, cover selection, and cover synthesis. The cover modification involves embedding the secret information directly into the cover media, typically by modifying the least significant bits of the cover. Cover selection, on the other hand, involves selecting a cover media that is most suitable for hiding the secret message. The cover media is chosen based on its ability to accommodate the secret information without significantly altering its visual or auditory quality. Finally, cover synthesis involves creating a new cover media specifically to hide secret information.

The most commonly used steganography techniques involve image files, as they are the most easily accessible and widely used media type. Earlier steganography in image files is accomplished by modifying the least significant bits of the pixels in the image. By changing the values of the least significant bits, the color of the pixel is changed slightly, which is often imperceptible to the human eye. Changing these values in a particular pattern can encode the hidden message into the image.

While steganography can be an effective means of hiding sensitive information, it is not foolproof. There are techniques that can be used to detect the presence of hidden information, such as statistical analysis of the cover media.

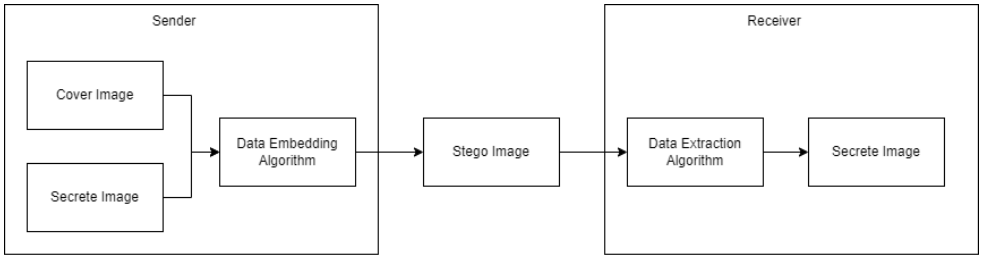


Figure 1.1: *Basic Process of Traditional Image Steganography*

Additionally, steganography can be vulnerable to attacks such as steganalysis, which involves analyzing the cover media to detect signs of hidden information. Therefore, steganography should always be used in conjunction with other security measures such as encryption to provide a more robust level of protection for sensitive data.

Steganography by cover modification is a commonly used technique in traditional image steganography, which involves embedding a secret message within a cover image using a modification algorithm. The modified image, known as a stego-image or carrier image, is intended to appear visually identical to the original cover image to the naked eye, while simultaneously concealing the embedded information in a way that cannot be easily detected by statistical analysis as in Figure 1.1.

The process of steganography by cover modification typically involves selecting a cover image that has certain characteristics, such as a high resolution or a complex pattern, that make it suitable for hiding the embedded information without significantly altering the visual appearance of the image. The message to be hidden is then encoded in a way that is designed to minimize any detectable changes to the cover image. The Modification to the least significant bits of the pixels in the image can be done in a variety of ways, such as using a predetermined pattern or randomly selecting pixels to modify.

Several techniques can be used to detect the presence of hidden information in a stego-image, including visual inspection, statistical analysis of the image, and the use of specialized software designed to detect steganography.

Cover modifications based image steganography methods are commonly classified into two main categories: spatial domain steganography and transform domain-based steganography (M. Liu et al. 2017). Spatial domain-based methods involve encoding the secret message at the least significant bits of the cover image while transforming domain-based methods modify the statistical features of the host image data.

However, these methods can subject the cover image to modification during information embedding, which can make them detectable by steganalysis tools. To

overcome this, researchers have proposed Coverless Image Steganography (CIS), which does not use a designated cover image. CIS includes methods such as steganography by cover selection and cover generation.

Steganography by cover selection involves selecting an appropriate cover image that has specific characteristics suitable for hiding the secret message. This method can be more challenging than traditional steganography as it requires selecting a cover image that can accommodate the hidden message without significantly altering the visual quality of the image. The process of selecting a cover image involves analyzing the image to determine its characteristics and then selecting an appropriate image that meets the requirements for hiding the secret message. Once an appropriate cover image has been selected, the secret message is then embedded into the image using a steganographic algorithm. The resulting stego-image appears identical to the original cover image and can be transmitted without arousing suspicion.

However, cover selection based on CIS has some major drawbacks such as the difficulty in finding a suitable cover object, a significant increase in the image database size, and limitations in its embedding capacity (Q. Li et al. 2021).

On the other hand, steganography by cover generation involves generating a cover image specifically for the purpose of hiding the secret message. This method offers more control over the cover image and the ability to create a cover image that is optimized for the hidden message.

Steganography by cover generation has some limitations. One of the most significant drawbacks is the increased computational complexity and time required to generate the cover image compared to other steganographic techniques. This is because creating a new image as a cover involves various computational steps, such as image synthesis or manipulation, which demand additional resources and time. This increased computational overhead can pose challenges in scenarios where real-time or near-real-time steganographic operations are required, potentially impacting the practicality or efficiency of the technique.

In conclusion, cover modifications based image steganography is a widely used technique for concealing secret information within an image. While these methods can be effective, they are not foolproof and can be vulnerable to detection by steganalysis tools. Coverless image steganography methods offer alternative approaches to hiding secret information that can be more robust and less vulnerable to detection.

1.2 Motivation

The common limitations of traditional image steganographic methods include achieving high hiding capacity and ensuring security against steganalysis tools.

However, the utilization of Generative Adversarial Networks(GANs) in steganographic models has been shown to improve upon these limitations, resulting in higher rates of hiding capacity and improved security. Dharmawimala (2023) has extended the model of X. Liu et al. (2020) to input two gray scale images as secret images using GANs. The motivation of this research is to extend the model of Dharmawimala (2023) to address the problem of RGB colour secret multi-image steganography while improving the shortcomings of recovery accuracy and produce quality stego images, limitations found within existing multi-image steganographic models.

1.3 Project Aims & Objectives

This project aims to design a new coverless steganography method capable of successfully encoding multiple RGB colour images within one carrier image with the ability to resist steganalysis tools. To do so, we intend to achieve the following objectives:

- **Identify and analyze existing image steganography methods and their architectures.** An in-depth analysis of the existing image steganographic methods and their architectures is beneficial in understanding how various components are put together to build effective steganographic models, which pave the way to design the architecture of the proposed model.
- **Design and implement a GAN-based model that is capable of the encoding and decoding process in multi-image steganography.** We design and implement a coverless multi-image steganography model using the knowledge gained from the investigations into existing steganography models.
- **Evaluate the proposed model to determine its performance in achieving multi-image steganography.** The performance of the proposed model will then be evaluated according to the evaluation plan described in section 3.7.

1.4 Research Gap

Coverless multi-image steganography enables encoding multiple secret images within a carrier image. The model proposed by Dharmawimala (2023) achieves the task of coverless multi-image steganography, however, a major shortcoming of this steganographic model is the ability to produce the carrier image with no visual information of the secret images. As depicted in Figure 1.2.

The utilization of GANs to build image steganography has proved to be effective in increasing the hiding capacity as opposed to state-of-the-art models. Coverless image steganography models based on GANs have achieved the added advantage of better security performance in addition to high hiding capacity.

Encoding multiple secret images in one carrier image implies that the model developed to do so must entail the ability to achieve high rates of hiding capacity as each secret image contains a lot of information. Due to the high hiding capacity offered by GAN-based image steganography models, we intend to develop a multi-image steganography model by extending GAN-based image steganography. To ensure further security, we explore the applicability of coverless GAN-based image steganography for RGB color multi-image steganography.

1.5 Problem Statement

Since existing coverless multi-image steganography methods, as mentioned by Dharmawimala (ibid.), are unable to generate stego images with RGB colour secret images that maintain a satisfactory level of accuracy and concealment to the human eye, there is a clear requirement for the development of a more efficient model to address this challenge. Considering the enhanced hiding capacity and imperceptibility achieved through the application of Generative Adversarial Networks (GANs) in steganographic models, we aim to explore the potential for enhancing coverless multi-image steganography specifically for RGB secret images by leveraging GAN technology.

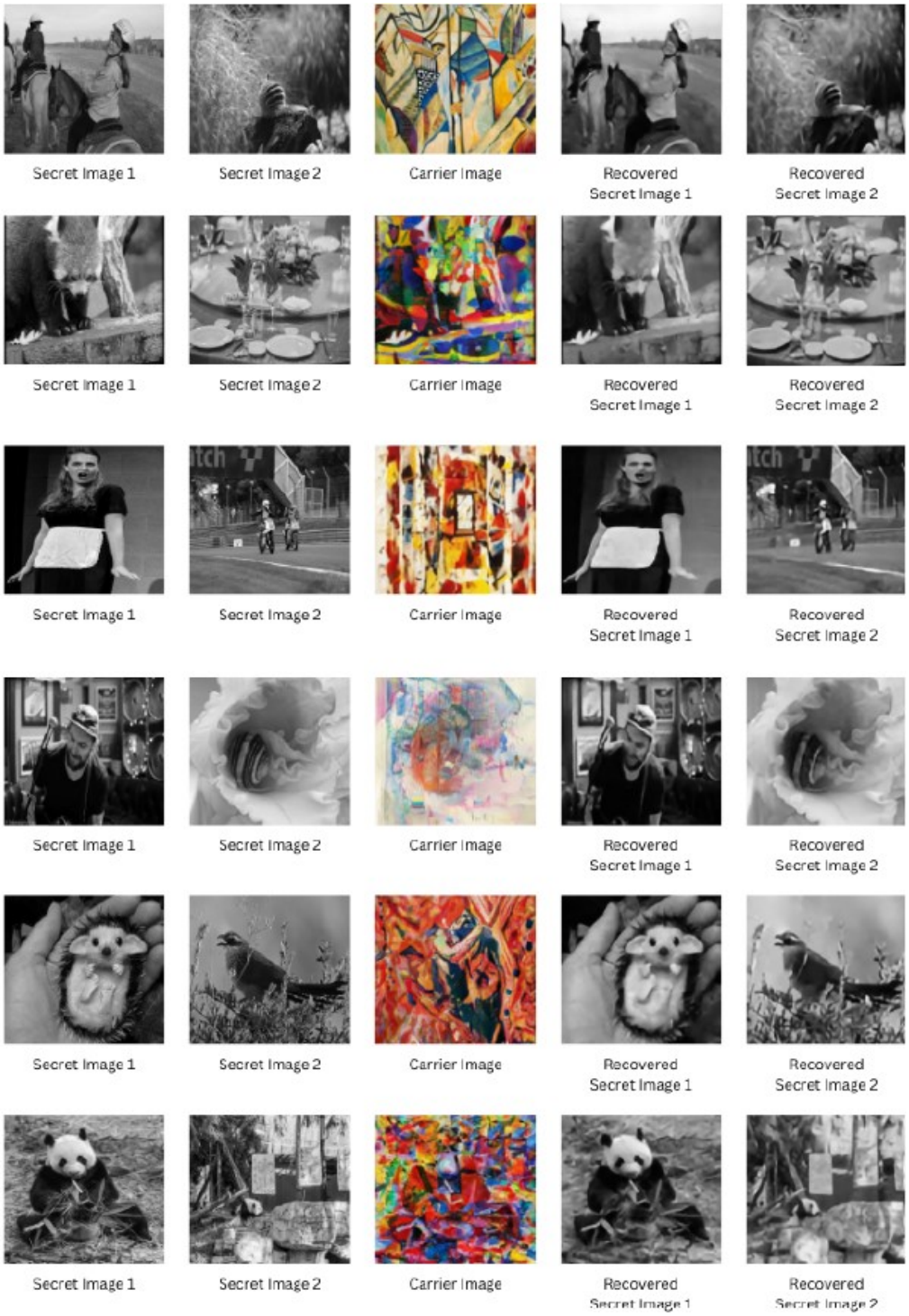


Figure 1.2: *Visual results of the proposed model of Dharmawimala 2023*

1.6 Research Question

To address the requirements of designing an effective coverless multi-image steganography method, as elaborated above, we intend to answer the following research question through the proposed study.

- **How can GAN-based coverless image steganography methods be extended to implement a model capable of multi-image steganography for RGB colour images?**

  When looking at the literature, it becomes evident that Generative Adversarial Networks (GANs) have been extensively employed to construct distinctive frameworks for the encoding and decoding phases of steganography techniques, as demonstrated in the works of H. Shi (2017), Volkhonskiy (2017), and Ruohan et al. (2019). Thus, our objective is to explore the feasibility and potential adaptations of GAN-based architectures to develop a coverless multi-image steganography model specifically designed for RGB colour images.

1.7 Significance Of The Project

Currently, there exist several effective steganographic models capable of successfully concealing information. However, despite recent advancements, multi-image steganography has not yet reached its full potential in terms of accuracy. Furthermore, to the best of our knowledge, no progress has been made in creating a coverless multi-image steganography model specifically designed for RGB colour images.

Hence, the focus of this research is to identify the limitations of current multi-image steganography methods and explore alternative approaches by leveraging GANs to develop a coverless multi-image steganography model for RGB color images. The development of such a model would make a significant contribution to the scientific community and find applications in various fields where information hiding is essential. Additionally, this model could serve as a foundation for the advancement of more sophisticated coverless multi-image steganography models for RGB colour images.

1.8 Research Methodology

This research project aims to explore the feasibility of applying advanced GAN-based image steganography techniques in the development of a coverless multi-image steganography model that offers enhanced recovery accuracy and security. The project will adopt a deductive approach to achieve its objectives. To begin with, an extensive literature review will be conducted to analyze the existing architectures of image steganography models. This review will be extended to examine the capabilities and applicability of current GAN-based image steganography models. By the end of this phase, a viable approach will be identified to extend GAN-based techniques for multi-image steganography, and an architecture will be designed to facilitate both the encoding and decoding processes of the coverless multi-image steganography model.

Once the architecture is established based on the findings of the initial literature review, the project will move on to the implementation phase. During this phase, the coverless multi-image steganography model will be developed, and an appropriate evaluation plan will be formulated to assess its capabilities. Publicly available datasets will be used to train the model, and test results will be obtained upon completion of the training phase. Finally, a comprehensive evaluation of the model will be conducted, drawing conclusions based on its performance and effectiveness.

1.9 Project Scope

This study will investigate the applicability of different Generative Adversarial Networks models in multi-image steganography for RGB colour images, Furthermore, during this study we will be designing and implementing a model capable of coverless multi-image steganography with the RGB secrete images. In addition, we will be training the proposed model and evaluating the model. This project will not investigate the applicability of other deep-learning models during the implementation. However, we will limit our study to designing a GAN based coverless multi-image steganography model for only two secret images.

1.9.1 Outline of the Dissertation

The subsequent sections of this dissertation are divided into seven chapters. This portion provides a summary of each chapter: Chapter 1 presents the introduction to and background of the study, alongside a concise overview of the entire dissertation. Chapter 2 expands on this foundation, engaging in a detailed exploration of the techniques and related work within the field. Chapter 3 presents the design

of the proposed methodology, leading into Chapter 4, which thoroughly describes the implementation of the proposed technique. Chapter 5 conducts a comprehensive evaluation of the approach, assessing its effectiveness and potential applications. The dissertation concludes with Chapter 6, summarizing the findings and suggesting avenues for future research.

Chapter 2

Literature Review

2.1 Steganography

Steganography is the practice of hiding secret information within seemingly innocuous communication. It has become a popular research topic due to the increase in multimedia content. The concept of steganography can be traced back to ancient times Johnson and Jajodia (1998), but its study within scientific literature began with the "Prisoners' Problem" (Simmons 1984), which illustrates the fundamental concept behind steganography and steganalysis.

Alice and Bob are two prisoners who want to come up with a plan to escape from jail. Unfortunately, all of their communications are monitored by a jail warden named Eve, who will punish them if she suspects they're exchanging information. Despite this obstacle, Alice and Bob are determined to hide their secret message in a seemingly innocent conversation, while Eve tries to detect any signs of suspicious communication. This scenario demonstrates the core idea of steganography and steganalysis. Alice and Bob's exchange of secret information represents steganography, while Eve's efforts to detect the hidden message is an example of steganalysis. This problem, modeled for steganography, is illustrated in Figure 2.1. In addition to uncovering secret information, steganalysis tools can also be used to determine the location and contents of the discovered secret information.

Steganography techniques have spread across different forms of communication, resulting in text steganography, audio steganography, image steganography, video steganography, and network steganography. However, in this context, we will narrow our attention to image steganography.

Image steganography involves using a cover image to conceal secret data, such as text or images, through various manipulations in order to remain undetected by steganalysis. Conversely, steganalysis is employed to inspect images and deter-

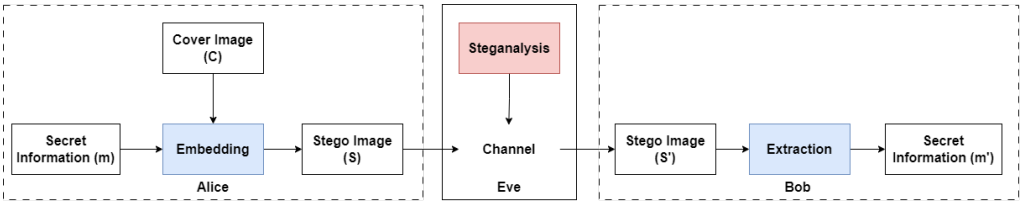


Figure 2.1: *Steganography and Steganalysis modeled in the Prisoner's Problem Simmons 1984*

mine whether they are normal images or stego-images that have been manipulated to hide secret data.

2.2 Image Steganography

In image steganography, a cover image is used to conceal the secret media in the form of text data or images by some form of manipulation with the aim of being undetectable by steganalysis. Moreover, steganographic schemes can be categorized further based on the construction of the cover image. The following section will delve into the three primary categorizations of steganography models.

2.2.1 Steganographic Methods

Image Steganography can be further classified into three categories based on the principles of constructing the carrier image as proposed by Fridrich (2010);

- Steganography by cover modification
- Steganography by cover selection
- Steganography by cover generation.

Subsequently, sections 2.2.1.1, 2.2.1.2, and 2.2.1.3 will provide an overview of the research conducted in each of these categories, respectively, while emphasizing their strengths and/or limitations.

2.2.1.1 Steganography by Cover Modification

Steganography by cover modification techniques involves manipulating an image to hide confidential information, and one common approach is to alter the least significant bit of each pixel to reflect a single bit of the secret message Hao Wu et al. (2005) and Mielikainen (2006). However, these methods have certain limitations, such as being sensitive to image manipulation and leaving behind visible

traces that could be detected by steganalysis algorithms. Adaptive steganography techniques aim to address these challenges by considering the statistical global features of the cover image, such as edge detection and texture analysis, to determine the optimal hiding locations for the secret data. By doing so, these techniques can enhance the security and robustness of image steganography, making it harder for attackers to detect and extract hidden information.

2.2.1.2 Steganography by Cover Selection

Coverless image steganography is a technique that does not modify a designated cover image, providing additional security. One approach to achieving coverless image steganography is steganography by cover selection, where a cover object is chosen based on a mapping between the image and the secret message. Several methods have been proposed, such as constructing an image database and selecting images with identical hash sequences to the secret message (Zhou, Sun, et al. 2015; Zhou, Q. Wu, et al. 2017), or using distortions to select the cover image with minimum total distortion (Y. Wang et al. 2020). However, these methods have limitations, such as low hiding capacity and the risk of deducing the mapping relationship between the hidden information and carrier image.

2.2.1.3 Steganography by Cover Synthesis

The third approach to steganography involves creating a cover image from the secret information itself. In the past, these methods produced unrealistic images. However, newer techniques utilize texture-based synthesis models (Wei 1999) or statistical feature models to generate more natural-looking cover images. Additionally, recent developments in Generative Adversarial Models have been used to create cover images that are well-suited for steganography. These methods offer the advantage of producing cover images that appear normal, making it difficult for steganalysis algorithms to detect the presence of hidden information.

2.2.2 Hiding Images Within Images

This section discusses how secret information can be hidden within images using steganography. Baluja (2017) proposed a model that uses an encoder-decoder network to hide an entire image within another image with minimal loss to both images. Later, Das (2021) introduced a multi-image steganography method that uses deep neural networks to encode multiple secret images within one carrier image, but this model results in a significant loss in the retrieved secret images.

Although the Karras, Laine, and Aila (2018), proposed architecture known as StyleGAN, is not a steganography model, it has achieved multi-image-to-image

synthesis to some extent. StyleGAN separates the image synthesis process into two stages: the generation of a latent vector that controls the overall image content and the transformation of this vector into an image with a specific style. The authors introduce a new technique called style mixing, which allows the manipulation of multiple style vectors to create a diverse set of images with different styles. While training StyleGAN is computationally intensive and requires a large amount of GPU memory, it provides greater control over image style.

However, StyleGAN is not designed to control the semantic content of the generated images, which means that the model may not always produce images that are consistent with a given input text or semantic description.

2.3 Steganalysis

Steganalysis is a critical tool for detecting and preventing the malicious use of steganography, which can be used for illegal activities like espionage and terrorism. The detection of steganographic content in images is a challenging task, as the hidden data is designed to be undetectable to the human eye and traditional security methods.

Deep learning models like GNCNN (Qian et al. 2015), Xu-Net (Xu, Hanzhou Wu, and Y. Q. Shi 2016), and Ye-Net (Ye, Ni, and Yi 2017) have been developed to enhance the accuracy of steganalysis by detecting steganographic content using convolutional neural networks. These models are trained on large datasets of images to recognize statistical patterns and irregularities that may indicate the presence of hidden information.

Grayscale images are commonly used in steganalysis as they provide a more straightforward representation of image data, and the absence of colour channels reduces the complexity of the models. These deep learning models generate probability vectors that indicate the likelihood of the image being a stego image or a normal image, allowing for the detection of steganography in digital media.

Modern steganography models such as Fu, F. Wang, and Cheng (2020), X. Liu et al. (2020), and Cao et al. (2020) use one or more of the steganalysis tools such as StegExpose, GNCNN and Xu-Net to test the security of the developed method.

2.4 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GANs) proposed in (Goodfellow et al. 2017) are a type of artificial intelligence that can create new, realistic images that resemble real-world images within a specific domain . GANs consist of two sub-models, the generator and discriminator, which work together during the training phase to

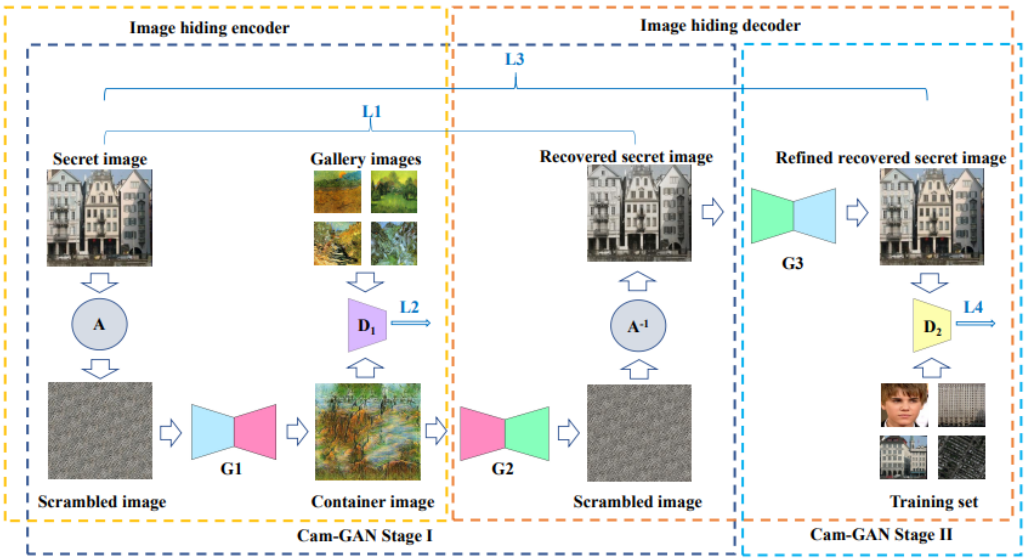


Figure 2.2: *The overall framework of Cam-GAN (X. Liu et al. 2020)*

create new image samples that closely resemble the training data. The generator produces an output image that is similar to the input image, while the discriminator classifies the generated image as real or fake.

2.4.1 Utilization of GANs in Steganography

Generative Adversarial Networks (GANs) have inspired new steganographic models that improve upon the limitations of traditional models. Traditional models face challenges in achieving high hiding capacity while ensuring high security, but GAN-based models have shown promise in addressing these limitations. Let's explore steganographic models that use GANs for information hiding and how they have improved upon the existing limitations.

2.4.1.1 Coverless Image Steganography using GANs

Duan (2018) introduced a novel CIS approach that employs a generative model. The method involves inputting a secret image into a database, which generates an independent image using a Wasserstein Generative Adversarial Network (WGAN) that differs from the original image. Another CIS method based on a DCGAN was proposed by Hu et al. (2018). This technique involves generating a stego image by feeding the relationship between the secret message and noise vector as input to the DCGAN.

K. A. Zhang (2019) proposed STEGANOGAN, which uses an encoder, decoder, and auxiliary critic network to hide information in carrier images. Cheddad et al. (2010) proposed a coverless steganography method based on the generation of anime characters using GANs. Mielikainen (2006) developed HIGAN, which hides an image within another image using an encoder, decoder, and discriminator network. Q. Li et al. (2021) presented a content-consistency CIS method to address the loss of detail in generated images. CycleGAN is an effective model for image translation that can be utilized in information hiding, according to Cui et al. (2019). Zheng (2019) proposed EncryptGAN, which is a steganographic model implemented as a Constrained-CycleGAN. Cam-GAN proposed by X. Liu et al. (2020) is the first coverless GAN-based image hiding model that achieves full-image size hidden capacity. In this model, a secret image undergoes a scrambling process. Subsequently, the scrambled image is passed through a generator to generate the carrier image. Upon receiving the carrier image, it is processed by another generator to unscramble the image, which is then fed into an additional generator to reconstruct the secret image. The complete structure of Cam-GAN is depicted in Figure 2.2. Remarkably, this model achieves a hiding capacity of 256 * 256 * 3 * 8 bits per image, demonstrating its ability to surpass the limitations of current steganographic models.

The model proposed in Dharmawimala (2023) is the first one to achieve coverless GAN-based image steganography for multi-image steganography for two grayscale images using GANs. In this model, secret images are sent through generator G1 to create an intermediate image. Then, the intermediate image is scrambled and sent through generator G2 to create the carrier image. At the receiver's end, the carrier image is sent through generator G3 to recover the scrambled image and reproduce the intermediate image. Then, it is sent through generator G4 to recover the secret images. The overall framework of the model is illustrated in Figure 2.3. This model achieves a hiding capacity of 256 * 256 * 8 * 2 bits per image, proving to overcome limitations within state-of-the-art steganographic models.

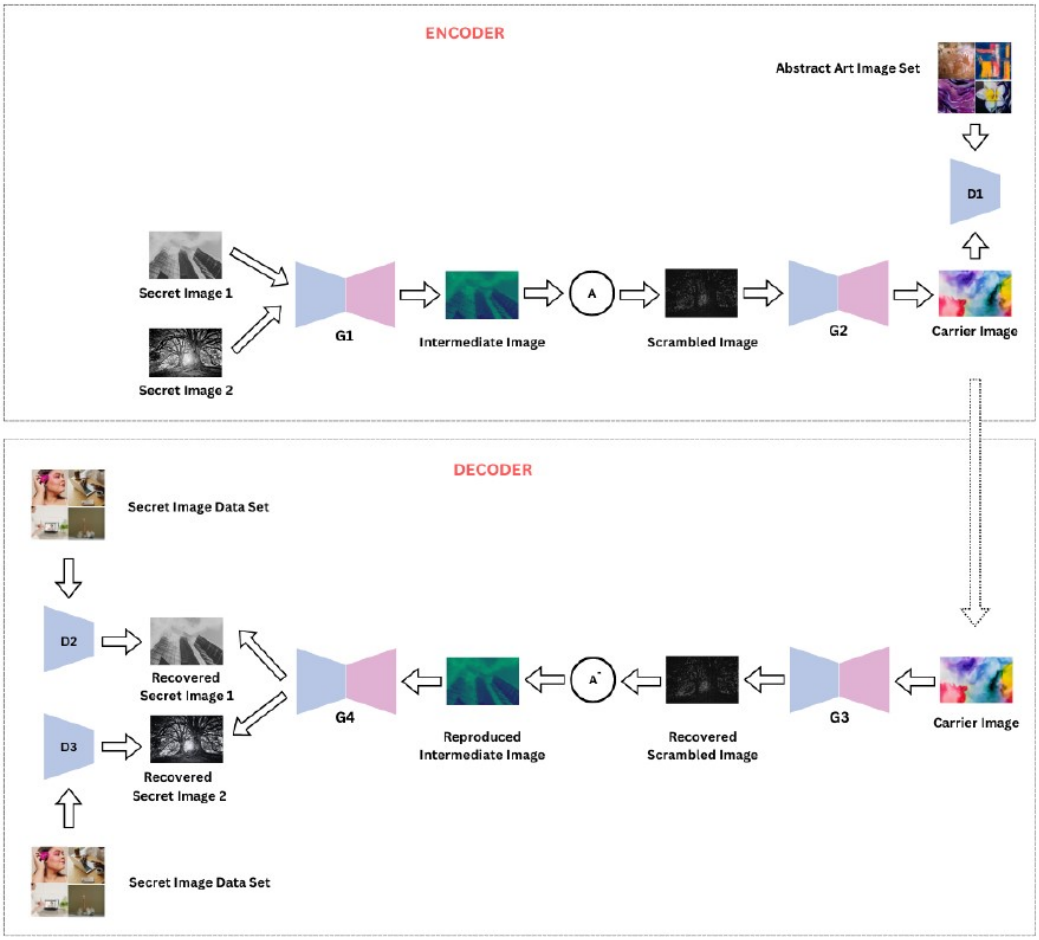


Figure 2.3: *The overall architecture of the proposed method Dharmawimala 2023*

Chapter 3

Design of The Proposing Model

The research design followed in this research consists of 7 main phases; Defining of Research Questions, Literature review, Data collection, Model design, Model implementation, Evaluation and Conclusion will be elaborated further within this section. Figure 3.1 illustrates the high level flow of these research phases along with their respective outcomes.

3.1 Define Research Questions

During this phase, we identified the research gap as stated in section 1.4 and the relevant research questions formulated are stated in section 1.6.

3.2 Literature Review

At the beginning of this project, We conducted a comprehensive literature review to find new methods that researchers have used for image steganography. This helped us understand the benefits and drawbacks of each approach. Based on our findings, We were able to create the model's architecture and develop an evaluation plan.

3.3 Data Collection

To test and train the proposed multi-image steganography we require two datasets:

- Secret Image Dataset
- Carrier Image Dataset

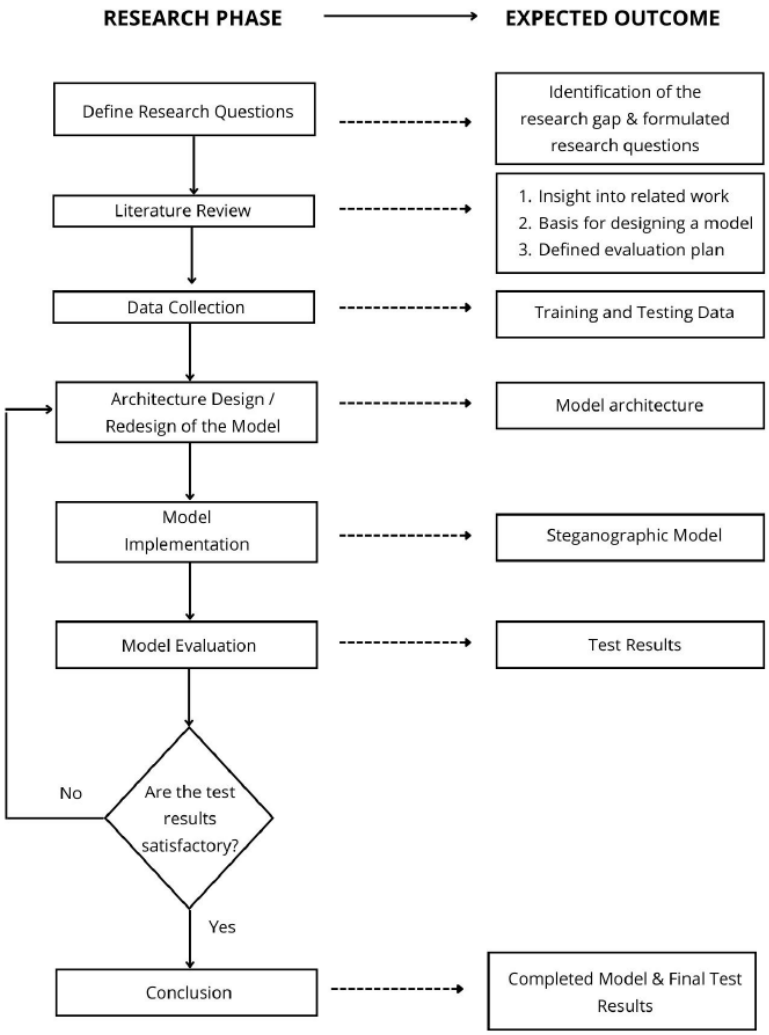


Figure 3.1: *Research Design*

We use the ImageNet [1] dataset(Russakovsky et al. 2015), provided by Princeton University and Stanford University, as the source of the secret images. This publicly available dataset was utilized by selecting 9,200 images for training and 1,000 images for testing. To generate abstract art as the carrier images for the proposed coverless multi-image steganography model, we used the Delaunay [2] dataset (Gontier, Jordan, and Petrovici 2022), which contains abstract and non-figurative art images. This dataset is also publicly available, and we use 2,500 images for training and 1,000 images for testing.

[1] `https://imagenet.org/`

[2] `https://github.com/camillegontier/DELAUNAY_dataset`

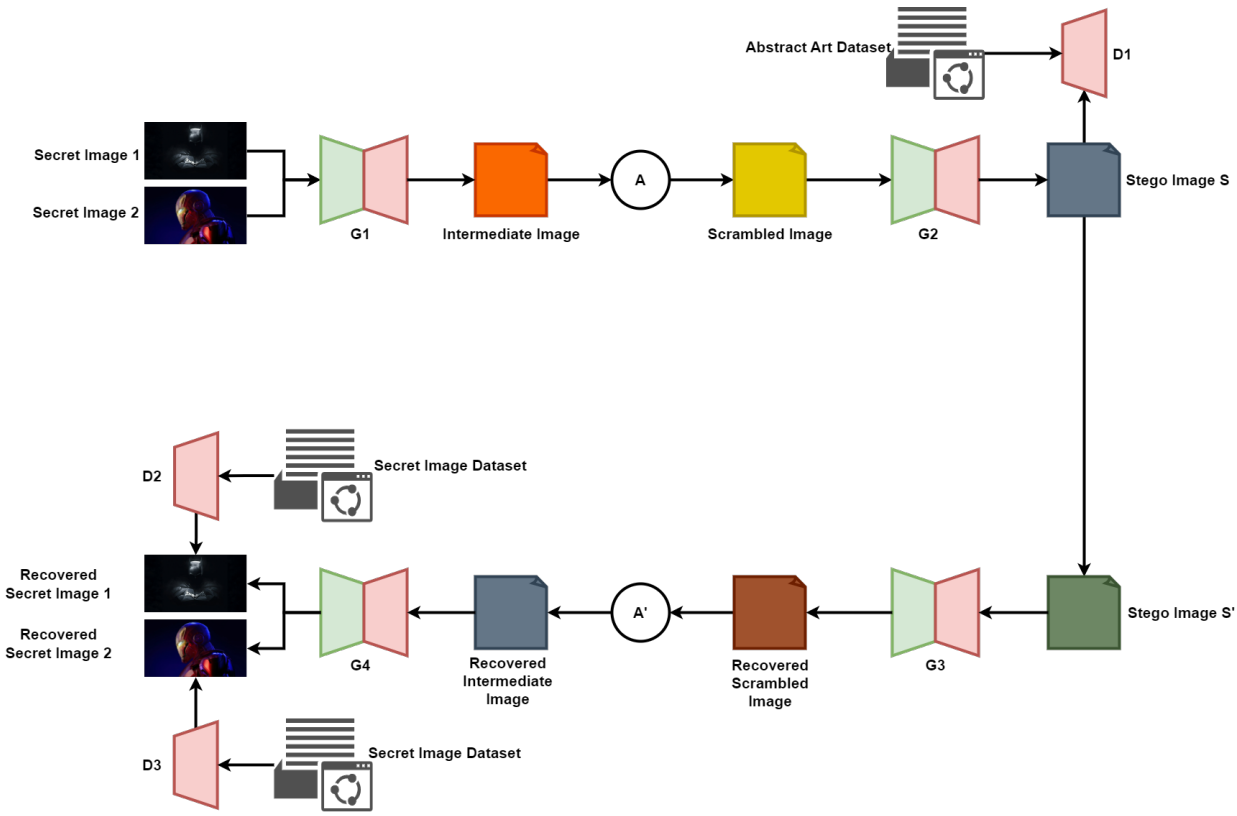


Figure 3.2: *The overall architecture of the proposed method*

3.4 Architecture Design

This section will discuss the initial architecture design of the novel approach proposed for coverless multi-image steganography based on GAN. Inspired by the framework 'Cam-GAN' proposed by X. Liu et al. (2020) and the work done by Dharmawimala (2023), we propose the model architecture illustrated in Figure 3.2.

In the process of encoding, the initial step involves passing two confidential images through the generator G1. This generator creates a combined representation of both images. Subsequently, an image scrambling technique is implemented on this intermediary image to conceal the overall specifics of the confidential content. The resultant scrambled image undergoes processing by generator G2, resulting in the creation of a carrier image, also known as a stego image. Furthermore, the integration of a discriminator, D1, comes into play to enhance the difficulty of distinguishing the carrier image from the general abstract art dataset, aligning with the typical architecture of Generative Adversarial Networks (GANs).

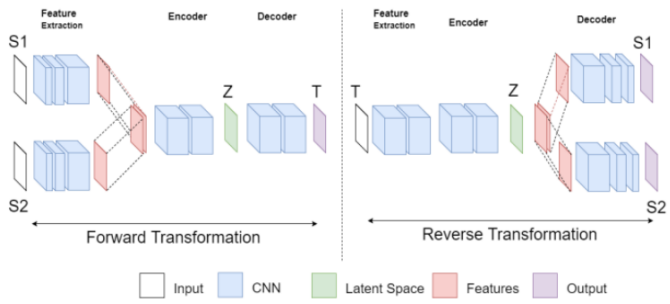


Figure 3.3: *The overall architecture for two input modalities of In2I (Perera, Abavisani, and Patel 2017)*

For the decoding process, the carrier image is transformed by generator G3, which in turn generates the restored scrambled image. An inverse transformation is then employed on this restored scrambled image to retrieve the intermediary image. Subsequently, this intermediary image is channelled through generator G4, which accurately reproduces the initial two confidential images. To add an extra layer of complexity and make it demanding to differentiate between the restored confidential images and the training dataset of confidential images, two discriminators, namely D2 and D3, are brought into operation.

3.4.1 Component Design

This section describes the network architecture of each component introduced in the overall model architecture.

3.4.1.1 Generator - G1

The generator G1 takes in two images and produces a fused representation of both images, which is referred to as the intermediate image. For this purpose, we adopt the forward transformation generator implemented in the In2I - Multi-image to image translation model (Perera, Abavisani, and Patel 2017). This generator can combine two images from different modalities into one image in a specified domain. The overall generator architecture for the In2I model is illustrated in Figure 3.3. This generator uses two feature extractors for each image and fuses the extracted features, which are, in turn, fed to the encoder. Then the encoder transforms it into a latent space. Lastly, the decoder generates a single image on this latent space.

The network architecture details of the two input modality forward transformation generators are shown in Figure 3.4.

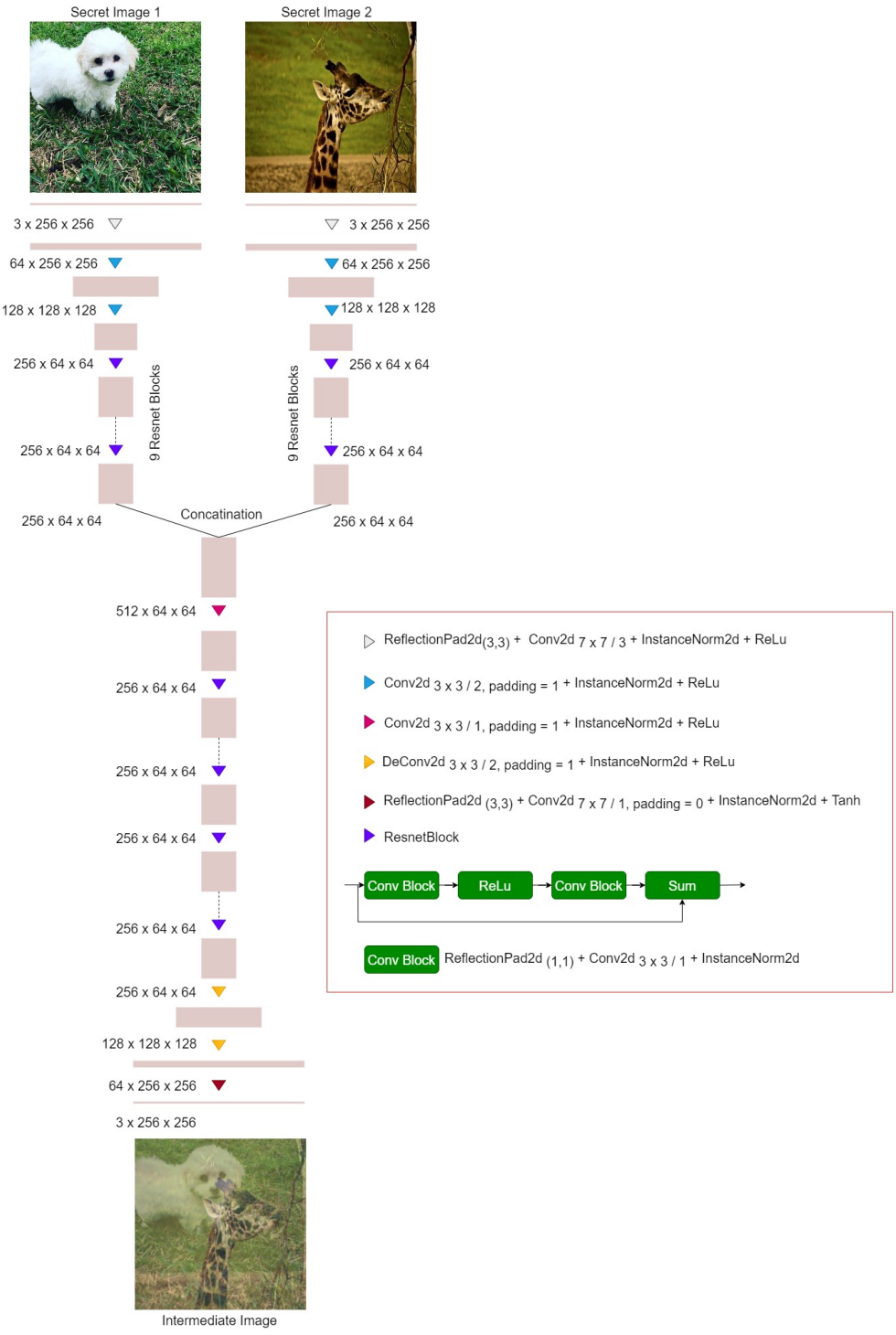


Figure 3.4: *Network architecture of G1*

3.4.1.2 Image Scrambling Transformation - A, A'

The encoder applies a transformation A to the intermediate image to obscure the global structure of the secret image, before passing it through the generator G2. In the decoder, the scrambled image is restored by applying the inverse transformation A' to produce the recovered intermediate image.

3.4.1.3 Generator - G2, G3

To create a carrier image for steganography, we use Generator G2 to convert a scrambled image into abstract art. To transform the carrier image back into the scrambled image, we use Generator G3. For this, we use the generator architecture from Cycle-GAN (K. A. Zhang 2019). The network architecture for both G2 and G3 can be found in Figure 3.5.

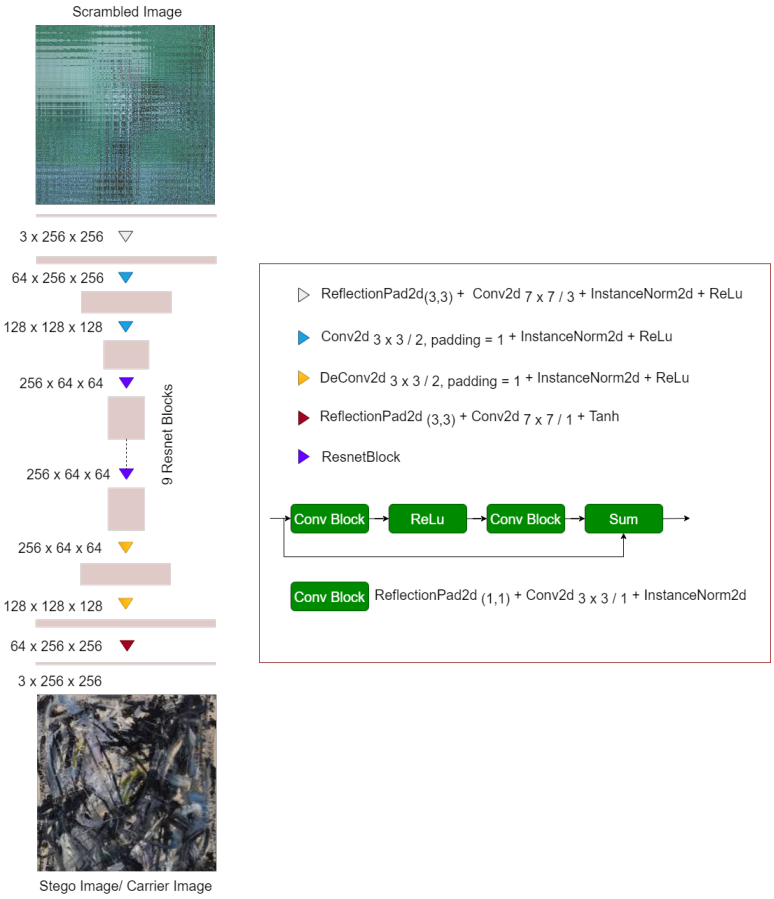


Figure 3.5: *Network architecture of G2 and G3*

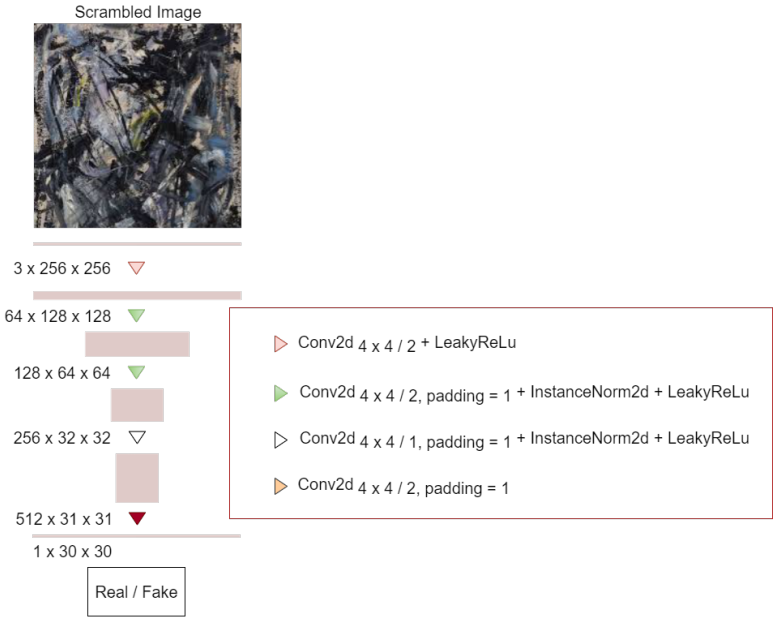


Figure 3.6: *Network Architecture of D1, D2 and D3*

3.4.1.4 Discriminator - D1, D2, D3

In order to seamlessly integrate the carrier image into an abstract art dataset, we utilize Discriminator D1. Additionally, we leverage Discriminators D2 and D3 to ensure that the recovered secret images align with the secret image dataset. To achieve our goals, we adopt the PatchGAN network architecture (Isola et al. 2016), which is also employed in the In2I model. For a visual representation of the network architecture of D1, D2, and D3, please refer to Figure 3.6.

3.4.1.5 Generator - G4

The G4 generator employs the recovered intermediate image to produce the covert images. This is accomplished through the utilization of the In2I - Multi-image to image translation model's reverse transformation generator(Perera, Abavisani, and Patel 2017). This generator can transform a single image into multiple images across diverse modalities. Refer to Figure 3.7 for a detailed overview of the network specifications for the reverse transformation generator's two input modalities.

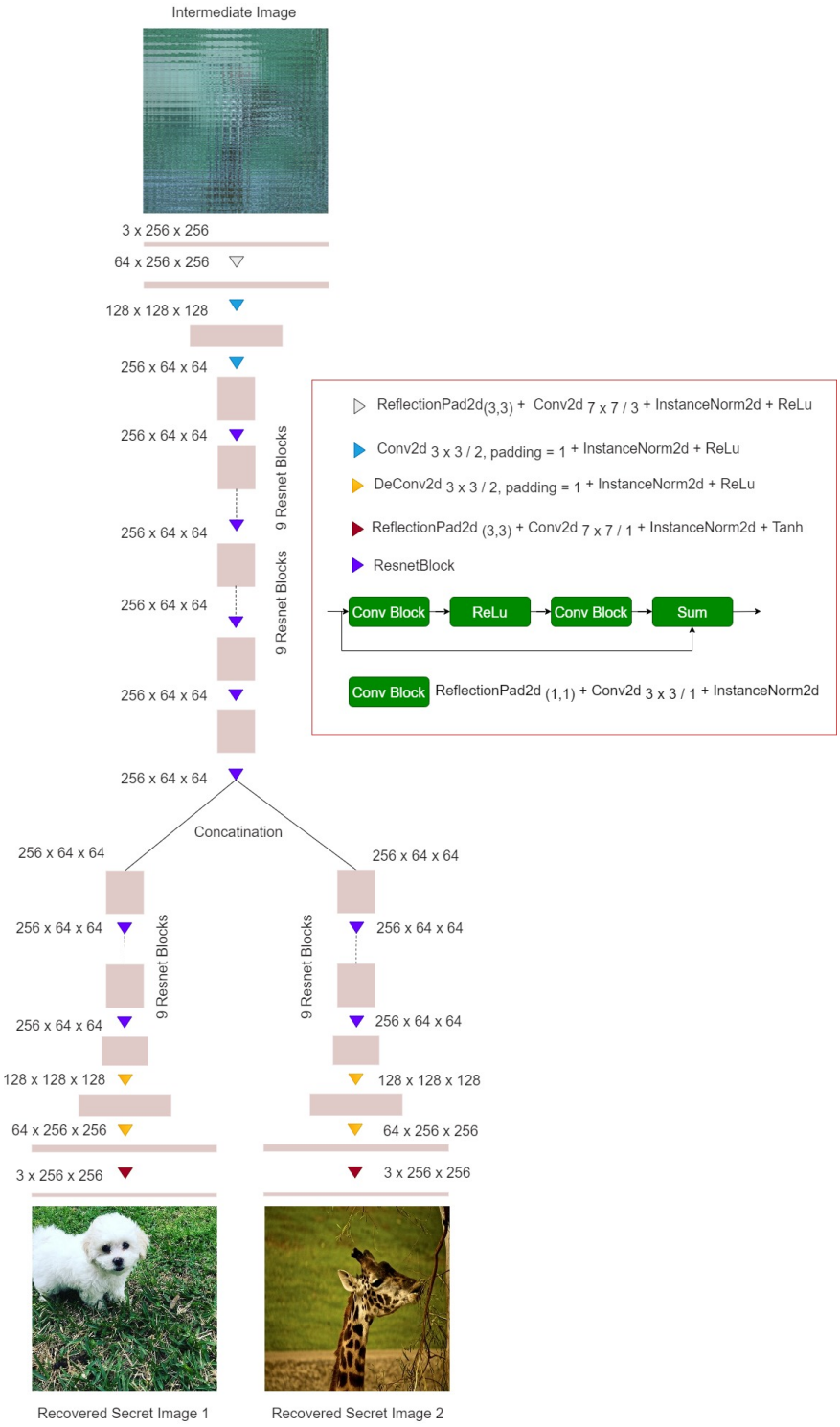


Figure 3.7: *Network architecture of G4*

S	Secret Image Domain
T	Carrier Image Domain
R	Intermediate Image Domain
Z	Latent Space
x	A data sample from an arbitrary domain X
$f_{X \longrightarrow Y}$	Transformation from domain X to Y
$h_{X \longrightarrow Z}$	Transformation between domain X and latent space Z
$g_{X \longrightarrow R}$	Transformation between domain X and intermediate domain R
$k_{R \longrightarrow X}$	Transformation between intermediate domain R and domain X
$P_{data(x)}$	Data distributions of domain X

Table 3.1: *Notations defined for the proposed model*

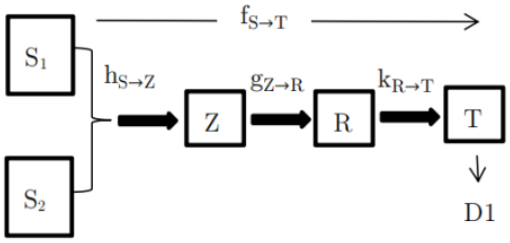


Figure 3.8: *Encoder Transformation of the proposed model*

3.5 Problem Formulation

The loss functions of the proposed model are formulated using the notations defined in Table 3.1. To make it easier to understand, a graphical representation of these notations is provided in Figures 3.8 and 3.9. Please note that the scramble transformation has been excluded from this illustration as it does not play a role in the formulation of the loss equation.

The objective of this model is to learn a transformation $f_{S \longrightarrow T}(.)$. Two images are fed as input to the encoder, and a single image is given as output. In a similar manner, the decoder $f_{T \longrightarrow S}(.)$ takes in one image as input and produces two images as output. Thereby, the objective of the proposed model can be depicted using the loss equations detailed below,

3.5.1 Adversarial Loss

We use the generator-discriminator pair: $f_{S \longrightarrow T}(.), D1(.)$ to learn the transformation $f_{S \longrightarrow T}$ (Perera, Abavisani, and Patel 2017). The data distributions of domain S and T are denoted $P_{data(s)}$ and $P_{data(t)}$ respectively. Here the generator attempts

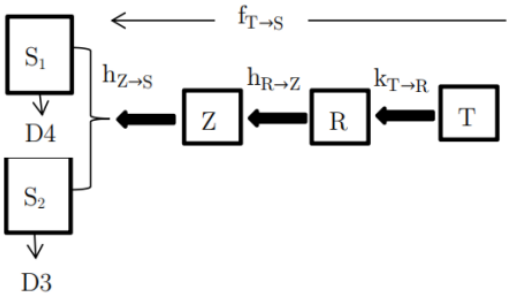


Figure 3.9: *Decoder Transformation of the proposed model*

to learn the $f_{S\longrightarrow T}$ transformation while the discriminator (D1) is trained to discriminate images from the generated images $f_{S\longrightarrow T}(s)$ from the target domain T. This process can be mathematically modelled by the adversarial loss as shown in Equation 3.1.

$$\mathcal{L}_{GAN,S\longrightarrow T} = E_{t\sim P_{data(t)}}[logD1(t)] + E_{s\sim P_{data(s)}}[1 - logD1(f_{S\longrightarrow T}(s))] \quad (3.1)$$

Similarly, we use a single generator to learn the transformation $f_{T\longrightarrow S}$ and two discriminators (D2 and D3), one for each secret image. Thereby, the adversarial loss for the decoder generator is depicted in Equation 3.2.

$$\begin{aligned}\mathcal{L}_{GAN,T\longrightarrow S} = & E_{s1\sim P_{data(s1)}}[logD1(s1)] \\ & + E_{s2\sim P_{data(s2)}}[logD2(s2)] \\ & + E_{s1\sim P_{data(s1)}}[1 - logD2(f_{T\longrightarrow s1}(t))] \\ & + E_{s2\sim P_{data(s2)}}[1 - logD3(f_{T\longrightarrow s2}(t))]\end{aligned} \quad (3.2)$$

3.5.2 Latent Consistency Loss

This loss was introduced in (Perera, Abavisani, and Patel 2017), and it was proposed that the adversarial loss alone is not enough to preserve the semantic information of the input images. The adversarial loss is only capable of ensuring that the generated image belongs to a certain target domain. Thereby, the latent consistency loss was introduced. This loss forces the equalization of the latent representation produced during the encoder transformation and the latent representation produced during the decoder transformation with respect to the same input. That is, for a given input image s, its corresponding latent representations $h_{S\longrightarrow Z(s)}$ are compared with the latent representations obtained from the decoder transformation $h_{R\longrightarrow Z}(k_{T\longrightarrow R}(f_{S\longrightarrow T}(S)))$ Thereby, the latent consistency loss

defined for the encoder transformation is presented in Equation 3.3.

$$\mathcal{L}_{Latent,S\longrightarrow R} = E_{s\sim P_{data(s)}}||h_{S\longrightarrow Z}(s) - h_{R\longrightarrow Z}(k_{T\longrightarrow Z}(f_{S\longrightarrow T}(S)))||_1 \quad (3.3)$$

Similarly, the latent consistency loss defined for the decoder transformation can be formulated as shown in Equation 3.4.

$$\mathcal{L}_{Latent,R\longrightarrow S} = E_{t\sim P_{data(t)}}||k_{T\longrightarrow R}(h_{R\longrightarrow Z}(t)) - h_{S\longrightarrow Z}(f_{T\longrightarrow S}(t))||_1 \quad (3.4)$$

3.5.3 Cycle Consistency Loss

By the cycle consistency loss, we force the transformation to have an accurate inverse transformation. The cycle consistency loss was initially introduced in CycleGAN (Chu, Zhmoginov, and Sandler 2017) and was also adopted in the In2I model (Perera, Abavisani, and Patel 2017). By the utilization of this loss in the proposed model we ensure that the reproduced secret images are similar to the secret images. The forward cycle consistency loss reflects the distance between the two input images and the two reconstructed images. Thereby, the forward cycle consistency loss can be modelled as depicted in Equation 3.5.

$$\mathcal{L}_{cyc,S\longrightarrow T} = E_{s\sim P_{data(s)}}[||f_{T\longrightarrow s}(f_{s\longrightarrow T}(s)) - s||_1] \quad (3.5)$$

Similarly, the reverse cycle consistency loss is modelled as shown in Equation 3.6.

$$\mathcal{L}_{cyc,T\longrightarrow S} = E_{t\sim P_{data(t)}}[||f_{S\longrightarrow T}(f_{T\longrightarrow S}(t)) - t||_1] \quad (3.6)$$

3.5.4 Intermediate Consistency Loss

The adversarial loss guarantees that the synthesized images belong to their respective domains. The latency loss ensures that the latent representation generated in the encoder transformation is similar to the latent representation generated in the decoder transformation. The cycle consistency loss assures that the reproduced secret images are similar to the original secret images. However, we have not explicitly given instructions to the model to ensure that the intermediate image in the encoder transformation is similar to the reproduced intermediate image in the decoder transformation. Thereby, we introduce an intermediate consistency loss to enforce the similarity of the intermediate image and the recovered intermediate image. This intermediate consistency loss defined for the forward transformation is described in Equation 3.7.

$$\mathcal{L}_{int,S\longrightarrow T} = E_{s\sim P_{data(s)}}||h_{S\longrightarrow Z}(g_{Z\longrightarrow R}(s)) - k_{T\longrightarrow R}(f_{S\longrightarrow T}(s))||_1 \quad (3.7)$$

The intermediate consistency loss defined for the backward transformation is presented in Equation 3.8.

$$\mathcal{L}_{int,T\longrightarrow S} = E_{t\sim P_{data(t)}}||k_{T\longrightarrow R}(t) - (g_{Z\longrightarrow R}(h_{S\longrightarrow Z}(f_{T\longrightarrow S}(t)))||_1 \quad (3.8)$$

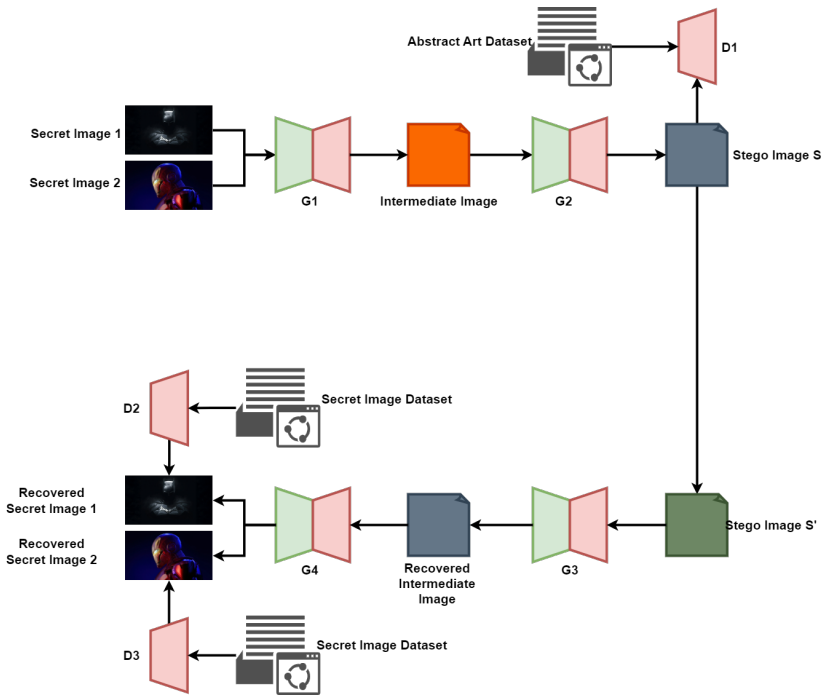


Figure 3.10: Model Architecture of Experiment 01

3.5.5 Cumulative Loss

The total of these four losses defined can be summed to obtain the final objective function. Thereby, the total loss/ cumulative loss formulated for the proposed model is presented in Equation 3.9.

$$\begin{aligned}\mathcal{L}_{total} = \mathcal{L}_{GAN,S\longrightarrow T} + \mathcal{L}_{GAN,T\longrightarrow S} \\ + \lambda_1(\mathcal{L}_{cyc,S\longrightarrow T} + \mathcal{L}_{cyc,T\longrightarrow S}) \\ + \lambda_2(\mathcal{L}_{Latent,S\longrightarrow R} + \mathcal{L}_{Latent,R\longrightarrow S}) \\ + \lambda_3(\mathcal{L}_{int,S\longrightarrow T} + \mathcal{L}_{int,T\longrightarrow S})\end{aligned} \tag{3.9}$$

3.6 Experimental Approach

This section describes a series of experiments undertaken for the evolution of the proposed model design.

3.6.1 Experiment 01

As explained in section 3.4, we built upon the existing 'Cam-GAN' model developed by X. Liu et al. (2020) and the research carried out by Dharmawimala (2023)

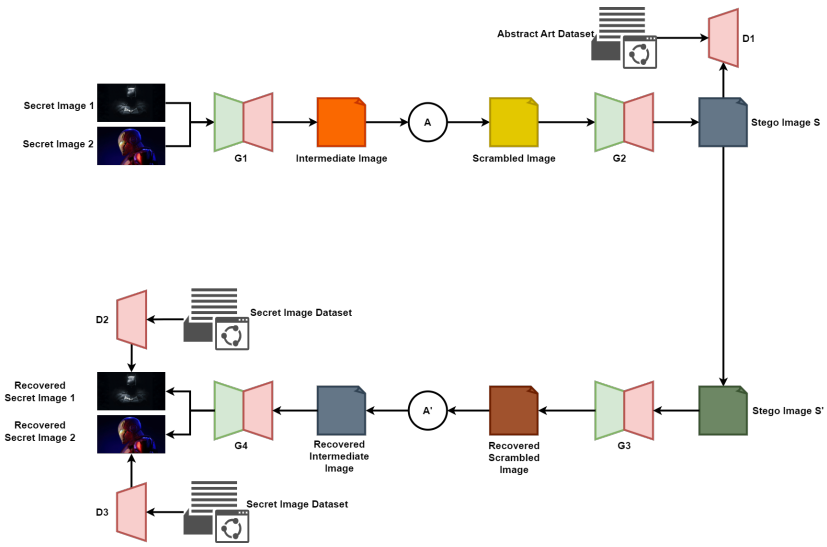


Figure 3.11: Model Architecture of Experiment 02

to create our proposed model. Our extension to the work of Dharmawimala (2023) allows it to accommodate Coverless RGB Multi-Image Steganography. To test the feasibility of this enhancement for multi-image steganography, we conducted experiment 01. As a first step, we implemented the architecture depicted in Figure 3.10.

In this experiment, two generators, G1 and G4, are used to process pairs of RGB images and produce corresponding RGB outputs. However, the image scrambling component necessary for image steganography is not included in this design, as the main objective is to explore the potential of extending Dharmawimala (ibid.) model towards RGB multi-image steganography. As a result, the global structure of the secret images will be preserved in the stego image during the initial experiments. This architecture served as the baseline design of the proposed model.

3.6.2 Experiment 02

In experiment 01, we managed to demonstrate that our model could reproduce images effectively. However, the outcome was a steganographic image that merged two secret images, signalling room for improvement. To address this, we launched a follow-up experiment to integrate an image scrambling mechanism. This step was crucial for investigating the potential of steganography that doesn't rely on a traditional cover image but can conceal multiple images. We chose to use an advanced version of the Arnold Transformation as our scrambling technique. Due to time constraints, we reduced the number of images in the dataset to train for more

epochs. The structure of the model we employed for this subsequent experiment is presented in Figure 3.11.

3.6.3 Experiment 03

Following the unsuccessful outcome of Experiment 02, we proceeded to Experiment 03, adopting the Arnold scrambler as detailed by M. Li, Liang, and He (2013). This method employs a straightforward coordinate transformation for a pixel's coordinates, as outlined in Equation 3.10.

$$\begin{bmatrix} \bar{x} & \bar{y} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} (mod\, m) \tag{3.10}$$

In this context, (x, y) represents the current coordinates of a pixel within the image, while $(\bar{x}, \bar{y})$ denotes the transformed coordinates. Given the RGB color model's usage, the maximum intensity value, m, is set to 256. Utilizing the inverse of this transformation, we can accurately restore the original pixel coordinates, ensuring that the process is reversible.

3.6.4 Experiment 04

During our analysis of the training process, we identified that, due to an oversight, computations were being performed on the CPU instead of the GPU. This was a significant bottleneck, given that tensor operations are substantially more efficient on GPUs. Acknowledging the opportunity for substantial enhancements in performance, we embarked on Experiment 04. The goal was to refine our model to fully exploit the computational prowess of GPUs. To evaluate the impact of this adjustment, we mirrored the setup of Experiment 01 with a dataset of 5000 images as the secret images. These were to be trained for 100 epochs, allowing us to directly compare the performance and efficiency improvements achieved through GPU optimization.

3.6.5 Experiment 05

With the adjusted model successfully producing the anticipated outcomes, we proceeded to incorporate a scrambling technique into our experiment. In this phase, we implemented a method where the fused image was segmented into predetermined sizes of blocks. These segments were then shuffled using a random permutation approach. Crucially, the permutation sequence was secured as a secret key, ensuring that only recipients in possession of this key could accurately reassemble

and retrieve the secret images. This added layer of security emphasizes the necessity for the receiver to have the specific permutation string, thereby enhancing the steganographic integrity of our method.

3.7 Evaluation Plan

The proposed model will be evaluated in terms of the image quality of the generated images, the security of the carrier images and the hiding capacity of the overall model.

3.7.1 Image Quality Evaluation

Image quality is crucial in evaluating steganography methods. The goal is to preserve the original secret images in the recovered images. Evaluation is done subjectively and objectively. The metrics used are Mean Squared Error (MSE), RMSE (Root Mean Squared Error), peak signal-to-noise ratio (PSNR), and Structural Similarity Index (SSIM). These measures have been widely utilized in the evaluation of steganography models, as evidenced by the literature ofQ. Li et al. (2021), X. Liu et al. (2020), and Cao et al. (2020).

3.7.1.1 Mean Squared Error & Root Mean Squared Error

MSE is a measure of the average squared difference between two images, the original and the reconstructed. On the other hand, RMSE is the square root of the average of the squared differences between the original and the reconstructed image. A lower value of MSE/RMSE indicates higher image fidelity, which means that the reconstructed images are closer to the original images. Equations 3.11 and 3.12 present the mathematical formulas to calculate MSE and RMSE, respectively.

$$MSE = \frac{1}{HW} * \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} [I(i,j) - I'(i,j))]^2 \tag{3.11}$$

$$RMSE = \sqrt{\frac{1}{HW} * \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} [I(i,j) - I'(i,j))]^2} \tag{3.12}$$

Where I is the original secret image, I' is the recovered secret image, H and W are the dimensions of the image and (i,j) represents pixel coordinates.

3.7.1.2 Peak Signal-to-Noise Ratio

PSNR is a measure used to evaluate the ratio of signal to noise in an image. It compares the maximum power of a signal to the power of noise that affects the image's accuracy. In image steganography, PSNR is used to measure the difference between the original secret image and the recovered secret image. A higher PSNR value shows that the recovered secret image is more similar to the original secret image. Conversely, lower PSNR values represent more significant distortion or noise in the embedding process. Equation 3.13 provides the mathematical formula used to calculate PSNR.

$$PSNR = 10 \log_{10} \frac{255^2 HW}{\sum_{i=1}^{H} \sum_{j=1}^{W} [I(i,j) - I'(i,j))]^2} \tag{3.13}$$

3.7.1.3 Structural Similarity Index

SSIM is a metric that measures the structural similarity between original and reconstructed secret images. Luminance, contrast, and structure are taken into account. Higher SSIM values indicate better image fidelity, meaning that the reconstructed secret image is more similar to the original in terms of its structure and appearance. The mathematical equation to calculate SSIM is presented in Equation 3.14.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\alpha_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\alpha_x^2 + \alpha_y^2 + c_2)} \tag{3.14}$$

The variables μ_x and μ_y represent image patches x and y, respectively, while α_x and α_y are the standard deviations of the image patches x and y, respectively. α_{xy} is the covariance of the image patches x and y, and c_1 and c_2 are constants used for stability in the computation of the SSIM index. It is essential to use a combination of objective and subjective measures to fully evaluate the quality of image reproduction.

3.7.2 Steganalysis Evaluation

Initially, our research design incorporated the use of GNCNN (Boehm 2014) and Xu-Net (J. Liu et al. 2020) for steganalysis, aiming to assess security performance as described by (Dharmawimala 2023). However, subsequent detailed analysis of both GNCNN and Xu-Net steganalysis mechanisms revealed their incompatibility with our proposed methodology. This issue comes from the basic principle of our method, which is based on coverless image steganography. Because of this, the tools we mentioned before, GNCNN and Xu-Net, don't fit our needs.

3.7.3 Hiding Capacity

The term ”Hiding Capacity” in the context of image steganography models denotes the maximum amount of secret information that can be embedded into a cover image without noticeably lowering its quality or drawing the attention of potential intruders. This capacity is typically quantified in bits or bytes and stands as a critical benchmark for gauging an image steganography model's efficacy. Essentially, a model with a substantial embedding capacity is adept at concealing secret information within the cover image efficiently.

Although coverless image steganography models do not undergo direct embedding, the hiding capacity of such models describes the amount of information (number of bits) that the carrier(stego) image can represent Q. Li et al. (2021),X. Liu et al. (2020),Cao et al. (2020). This measure is represented in terms of bits per cover ($bits * cover^{-1}$). The obtained hiding capacity value will be compared against other state-of-the-art steganography models.

Chapter 4

Implementation of The Proposing Model

During this phase, we will implement the model designed in the previous phase. By the conclusion of this phase, we will obtain a fully implemented steganographic model.

4.1 Language and Implementation Tools

We chose Python 3 as the programming language for our project because of its extensive collection of libraries designed for deep learning and computer vision applications. Our study benefited from the use of popular open-source machine learning libraries, including PyTorch for deep learning frameworks, OpenCV for computer vision tasks, and NumPy for numerical calculations. Torch was also added to provide more deep learning capabilities. To assist the model's training and testing operations, we used WinSCP, a program that allowed us to create a secure SSH connection to the 'AntPC' server. This set of tools and libraries laid the foundation for the efficient development and implementation of our proposed model.

4.2 Experimental Setup

We used Python version 3.6.9 for all of our experiments, including model creation and assessment. These actions were performed on the 'AntPC' server, which was powered by four NVIDIA GeForce RTX 208 GPUs. This server runs Ubuntu GNU 18.04.1 LTS, is powered by an Intel E5-2620 v4 CPU clocked at 2.10GHz, and has 125.65GB of RAM. This system setup provides a reliable foundation for our study, allowing for fast processing and analysis.

4.3 Data Preprocessing

In this study, the secret images utilized were sourced from the ImageNet dataset, as outlined by (Russakovsky et al. 2015). This dataset was bifurcated into two subsets for the purposes of training and testing. Specifically, the training subset was comprised of 5000 images, and the testing subset included 1000 images. The carrier images, essential for the steganographic process, were procured from the Delaunay dataset, as documented by (Gontier, Jordan, and Petrovici 2022). Similar to the secret images, these were also segmented into training and testing groups, with the former containing 2500 images and the latter 1000 images.

To standardize the input, all images were resized to a uniform dimension of 256 x 256 pixels. The process involved vertically concatenating two secret images to create a singular input for the model under consideration. This methodological approach facilitated a controlled environment for training and testing the proposed steganographic model, ensuring that the input data was consistently formatted across all experiments.

4.4 Model Development and Training

This section thoroughly investigates the many aspects of our proposed model, delving into both the structural components and the experimental journey that created its final form.

4.4.1 Implementation Details of Model Components

The proposed architecture consists of two main parts: an encoder network and a decoder network. The encoder network is composed of generator G1, an image scrambler, and generator G2, and it is paired with discriminator D1. The decoder network is composed of generator G3, an image scrambler, and generator G4, and it is paired with discriminators D2 and D3. The details of each component are explained in this section. The model has a total of 60,059,793 parameters.

4.4.1.1 Generator - G1

As shown in section 3.4.1.1, the generator G1 is a complex component of our proposed model that combines two RGB images to create a single image that visually depicts a combination of the two inputs. G1 is implemented using five unique subcomponents: model1, model2, model_fusion, model_pre, and model_post. Below is an overview of the role and architecture of each sub-component:

- **model1 & model2**
  These first sub-components are responsible for processing the incoming images individually. model1 is dedicated to secret image 1, while model2 is responsible for secret image 2. Both models have a similar architecture, with three convolutional layers extracting features from input images and four ResNet blocks meant to enhance feature analysis without increasing network complexity too much. The outputs of model1 and model2 are then combined to generate a concatenated feature set including information from both hidden images.

- **model_fusion**
  This sub-component serves as a link between the initial feature extraction and the next processing phases. It consists of a single convolutional layer that combines the concatenated outputs from model1 and model2, guaranteeing that the fusion of features is properly prepared for the next step of transformation.

- **model_pre**
  Following model_fusion, the merged characteristics enter the model_pre network. This network has three ResNet blocks that refine the features and project them into a latent space. This latent space represents a compressed and encoded version of the input data, which is ready for the final transformation.

- **model_post**
  The model_post network handles the final phase in the generator G1 cycle. It is in charge of transforming the encoded characteristics from the latent space into a visible representation. The architecture of the model_post comprises two ResNet blocks, which aid in the preservation of image features during reconstruction. This is followed by two deconvolutional layers that upscale the images back to their original dimensions, as well as a final convolutional layer that modifies the features to generate the intermediate fused image.

Each of these sub-components is crucial to the complicated process of image fusion, exhibiting the delicate balance of feature extraction, modification, and reconstruction necessary to produce the desired result.

4.4.1.2 Image Scrambling Transformation - A, A'

The encoder applies a transformation A to the intermediate image to obscure the global structure of the secret image, before passing it through the generator G2.

In the decoder, the scrambled image is restored by applying the inverse transformation A' to produce the recovered intermediate image.

4.4.1.3 Generator - G2

In section 3.4.1.3, the generator G2 is explained as a critical component of the proposed steganography paradigm, meant to convert an intermediate image into an abstract representation that acts as the stego image. This abstract image is critical for concealing the encoded information inside the visual content, making it an essential component of the steganographic method. The architecture of generator G2 is precisely built to make this transition possible, ensuring that the resultant stego-image strikes a balance between abstraction and the underlying encoded information. Here is a breakdown of the generator G2's architecture.

- **Three convolutional layers**
  These early layers process the incoming intermediate image, extracting and refining features required for the later transformation process. Convolutional layers are excellent in detecting patterns and features in images, making them a common feature in image processing models.

- **Nine ResNet blocks**
  Following the convolutional layers, the model employs nine Residual Network (ResNet) components. ResNet blocks are intended to enable deeper networks by enabling gradients to flow through the architecture more efficiently, hence avoiding the vanishing gradient problem. These blocks aid in keeping the fundamental aspects of the intermediate image while making the required changes to reach the abstract representation.

- **Two deconvolutional layers**
  Deconvolutional layers, also known as transposed convolutional layers, are used to upscale feature maps, hence expanding the spatial dimensions of the processed image. This is critical for restoring the image to its target resolution following the extensive feature extraction and alteration procedure.

- **One convolutional layer**
  The final layer in generator G2 is another convolutional layer. This layer serves as a refining stage, fine-tuning the abstract image's features and ensuring that the resulting stego-image has the proper texture and look. It changes the characteristics to create a final image that resembles an abstract artwork while concealing the encoded information inside its patterns.

The carefully designed architecture of generator G2 is critical in attaining the intended result of creating a stego-image that not only effectively conceals the

secret information but also looks to the inexperienced eye as a harmless abstract image. The steganographic model's success depends on striking a balance between aesthetics and functionality.

4.4.1.4 Generator - G3

Generator G3 is crucial to the steganography model because it reverses the process applied by Generator G2, taking the stego-image, which resembles abstract art, and reconstructing the intermediate image from it. This feature is critical for retrieving encoded information from the stego-image, making G3 a necessary component for the decoding section of the steganography process. G3's architecture is identical to that of G2, allowing for the same efficiency and precision in reverse operations.

4.4.1.5 Generator - G4

Section 3.4.1.5 describes how to extract the two secret images from the recovered intermediate image using a generator made up of four critical sub-components: model, model_pre, model_post1, and model_post2. This structure is meant to handle the intermediate image in a sequential manner, first encoding it into a latent space representation and then decoding it back into the two original secret images. Below is a breakdown of each sub-component's job and architecture inside the generator:

- **model**
  This initial stage analyzes the recovered intermediate image using three convolutional layers to extract fundamental features. Following that, four ResNet blocks are used to analyze these features further, improving the ability of the model to detect complicated patterns and relationships inside the image. The result is a latent space representation, which is a compressed and encoded version of the image's key information.

- **model_pre**
  Using the latent space representation from the previous step, model pre employs three ResNet blocks to begin the process of decoding or translating the encoded information back into a visual format. This stage seeks to recreate a fused representation of the two original secret images in order to separate them in the next phases.

- **model_post1 & model_post2**
  These sub-components are responsible for the recovery process's final phase,

which is to break the fused representation into two independent hidden images. Models post1 and post2 have a similar architecture, with two ResNet blocks to refine the features further, two deconvolutional layers to upscale the image dimensions and a final convolutional layer to adjust the image features to ensure accurate reconstruction of the original secret images. Model post1 is devoted to creating secret picture 1, whereas model post2 is focused on reconstructing secret image 2.

The sequential flow of these sub-components assures a thorough process of feature extraction, encoding, and decoding, which results in the correct recovery of the original secret images. This structure not only emphasizes the generator's complexity and efficiency but also the advanced methodology used to accomplish exact image recovery in the steganography model.

4.4.1.6 Discriminator - D1, D2, D3

The discriminator components D1, D2, and D3 perform separate functions within the steganography model, each focusing on a different part of the image generation and verification process, as described in section 3.4.1.4. The usage of the PatchGAN network across these discriminators is a deliberate decision intended to improve the model's capacity to successfully distinguish between different types of images. Let's take a deeper look at the responsibilities of these discriminators and the architecture of the PatchGAN network they use:

- **Discriminator D1**
  Focuses on distinguishing between genuine abstract art and abstract images created by the steganography model. Its fundamental purpose is to guarantee that the created stego images closely resemble actual abstract art, hence increasing the concealment of the steganographic process.

- **Discriminator D2 & D3**
  These are responsible for discriminating between the generated images and the original secret images. Their goal is to confirm the integrity of the secret images reconstructed by the model, guaranteeing that the decoding process successfully recovers the original material without significant loss or distortion.

- **PatchGAN Architecture**
  The PatchGAN network was chosen because it is more successful at evaluating images at the patch level than at assessing the complete image. This method enables more detailed and subtle discrimination by focusing on the texture and style of local imagine patches. The architecture comprises of five convolutional layers, which are briefly detailed below:

- **First Layer**
  Extracts basic features from the input image, such as edges and simple textures. This layer usually runs at a higher resolution to ensure accurate feature extraction.
- **Middle Layers**
  These layers gradually enhance the network's depth, extracting increasingly complicated and abstract features from images. With each layer, the network explores deeper into the image's content, detecting patterns and traits that distinguish genuine from generated images.
- **Last Layer**
  Combines the retrieved features into a collection of outputs that the network utilizes to make discriminating decisions. During this stage, the network evaluates the accumulated features to determine the legitimacy of the image patches.

The implementation of PatchGAN in D1, D2, and D3 emphasizes the model's emphasis on deep and exact image analysis, guaranteeing that the generated images are not only visually appealing but also survive detailed study. This level of selectivity is critical for the steganography model's efficacy and security it has a direct impact on the model's capacity to generate and retrieve secret images that are indistinguishable from their authentic counterparts.

4.4.2 Implementation Details of Conducted Experiments

4.4.2.1 Experiment 01

As previously presented in section 3.6.1, experiment 01 implements the following components as depicted in Figure 3.10,

- Encoder Generator - Generator G1 and G2
- Decoder Generator - Generator G3 and G4
- Discriminators - D1, D2, D3

The required time of training for this experiment was approximately 1 day 19 hours and 12 minutes, and the implementation details and parameter values utilized for this experiment are depicted in Table 4.1.

Parameter	value
Size of the image	256*256
Number of images	9162
Number of channels	3
Learning rate	0.0002
Number of epochs	20
λ_1	10
λ_2	1
λ_3	10
Image scrambler	False

Table 4.1: Implementation details of the Experiment 01 training process

Figure 4.1: Reference image before and after the Improved Arnold transformation.

4.4.2.2 Experiment 02

An image scrambling component has been developed using the Improved Arnold transformation. However, in experiment 02, the stego image generation process failed to generate an abstract image. Furthermore, the recreated images did not contain any information from the original secret images. Figure 4.1 depicts a reference image before and after the Improved Arnold transformation.

The required time of training for this experiment was approximately 14 days and 15 hours, and the implementation details and parameter values utilized for this experiment are depicted in Table 4.2.

Parameter	value
Size of the image	256*256
Number of images	5000
Number of channels	3
Learning rate	0.0002
Number of epochs	100
λ_1	10
λ_2	1
λ_3	10
Image scrambler	True

Table 4.2: Implementation details of the Experiment 02 training process

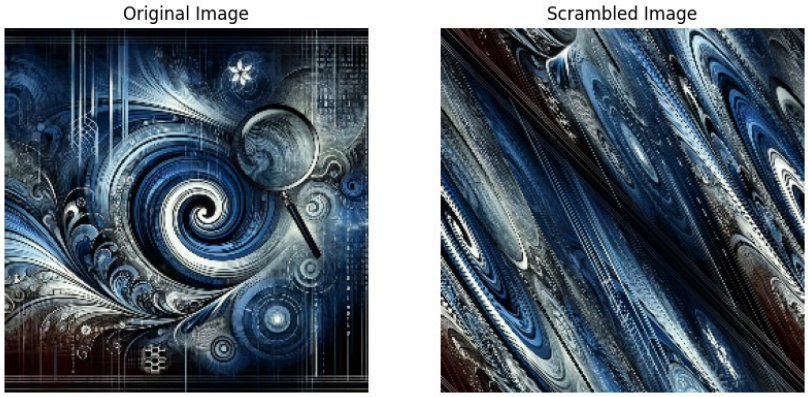


Figure 4.2: Reference image before and after the Arnold transformation.

4.4.2.3 Experiment 03

In Experiment 03, we implemented the Arnold transformation, as introduced by M. Li, Liang, and He (2013), as our chosen method for data scrambling within the scrambling component of our study. The implementation details for this experiment were directly aligned with those outlined in Experiment 02, particularly concerning the training process. Figure 4.2 depicts a reference image before and after the Arnold transformation.

4.4.2.4 Experiment 04

As detailed in Section 3.6.4, we implemented structural modifications to the complete model to leverage GPU capabilities for both processing and training phases. The implementation details and parameter values utilized for this experiment are

Parameter	value
Size of the image	256*256
Number of images	5000
Number of channels	3
Learning rate	0.0002
Number of epochs	100
λ_1	10
λ_2	1
λ_3	10
Image scrambler	False

Table 4.3: Implementation details of the Experiment 04 training process

depicted in Table 4.3. This approach was taken to ensure that any observed differences in performance or outcomes could be attributed directly to the utilization of GPU processing capabilities, rather than variations in experimental setup or parameter configurations.

4.4.2.5 Experiment 05

As detailed in Section 3.6.5 we have divided the fused image into 4 equal size blocks and shuffled them using the random permutation. Figure 4.3 depicts a reference image before and after the shuffling process. Then the same permutation sequence is provided as an input to the receiver end parallel to the steg image as a key to recover the secret images. The implementation details and parameter values utilized for this experiment are depicted in Table 4.4.

Parameter	value
Size of the image	256*256
Number of images	5000
Number of channels	3
Learning rate	0.0002
Number of epochs	100
λ_1	10
λ_2	1
λ_3	10
Block shuffler	True

Table 4.4: Implementation details of the Experiment 05 training process

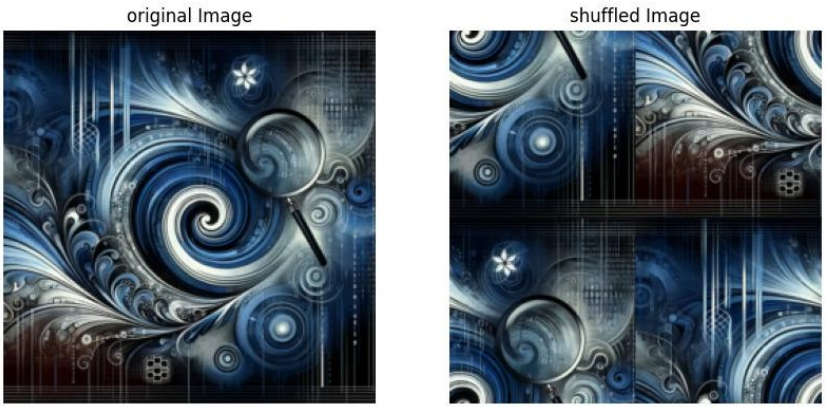


Figure 4.3: Reference image before and after the shuffling with 4 blocks

Chapter 5

Result and Evaluation

5.1 Result

5.1.1 Experiment 01

The visual results of this experiment are presented in Figure 5.1. By analyzing Figure 5.1, we can conclude that the desired outcome has been partially achieved. The carrier images represent a fused representation of the two secret images (As the image scrambling component has not been implemented), and this produced image resembles abstract art. Only one recreated image retains the original colour information, revealing the need for an image scrambler and model parameter improvement.

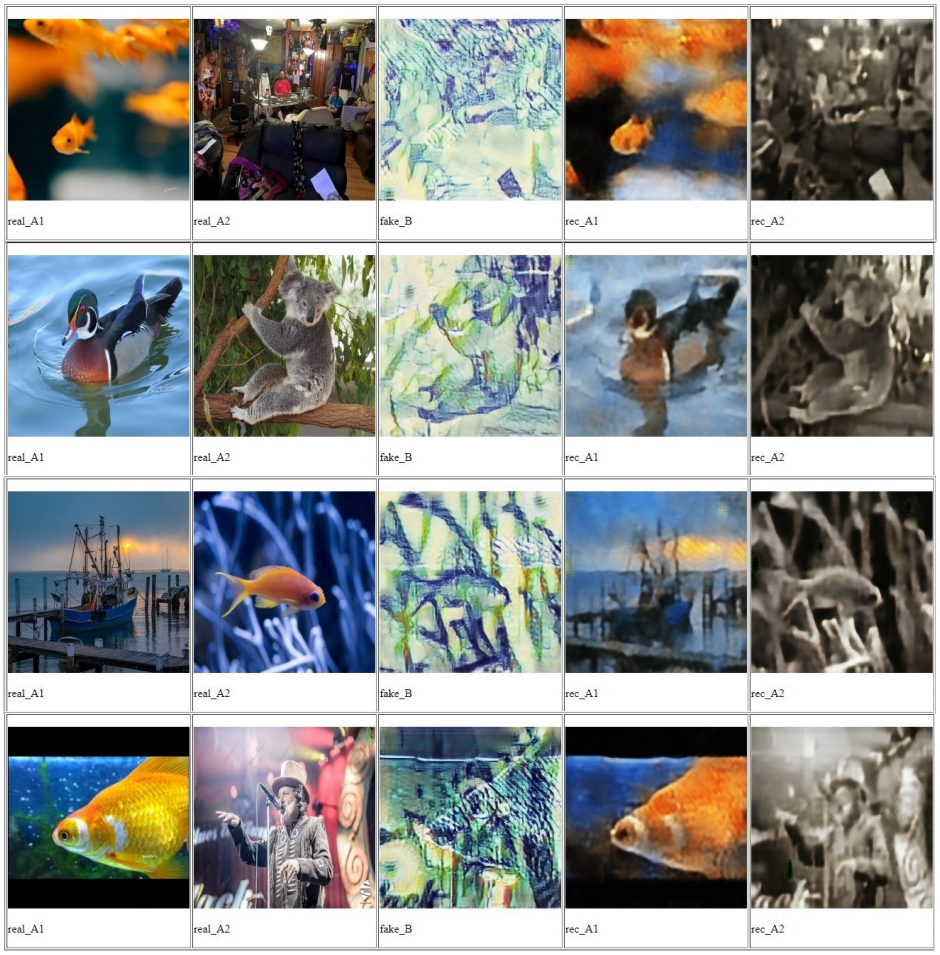


Figure 5.1: Visual results of experiment 01

5.1.2 Experiment 02

The visual results of this experiment are presented in Figure 5.2. Unfortunately, the steganography process was unsuccessful in generating an abstract image, and the recreated images did not contain any information from the original secret images. Following experiment 02, the use of the Improved Arnold Transformation scrambler resulted in intense scrambling, leading to the failure of stego image generation. Therefore, in future experiments, we must aim to improve the image scrambling model and enhance the accuracy of image recovery.

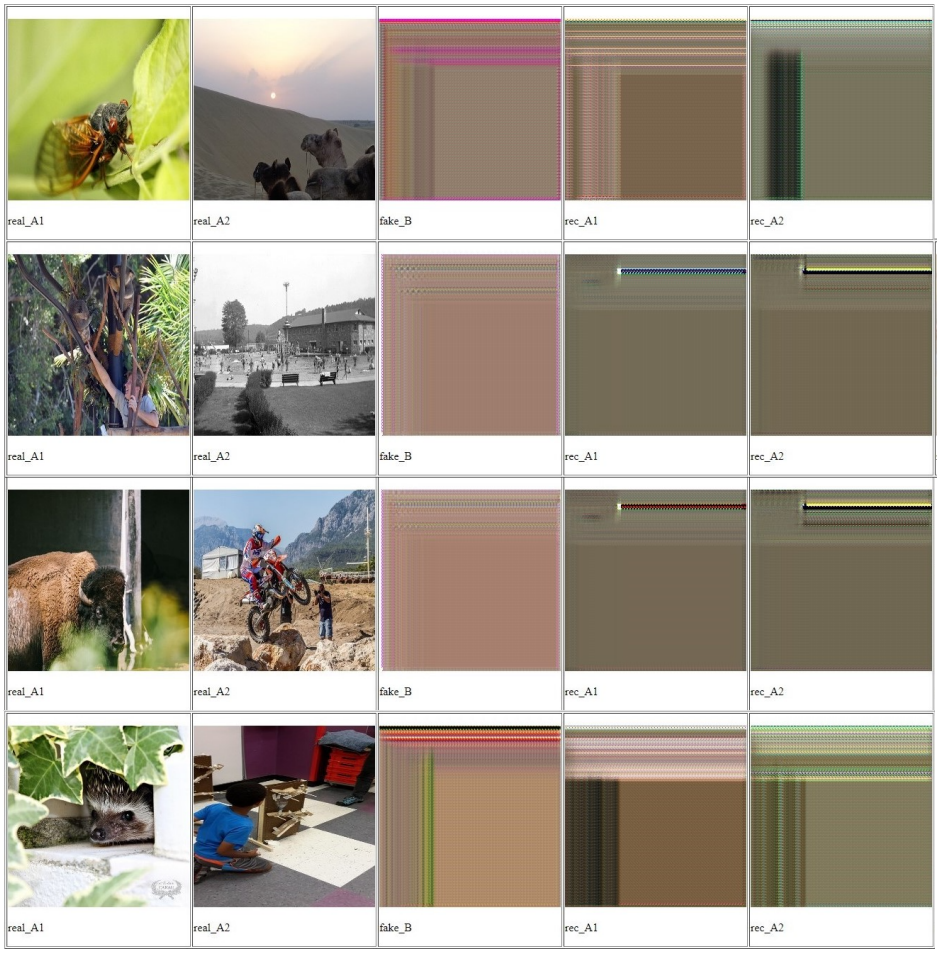


Figure 5.2: Visual results of experiment 02

5.1.3 Experiment 03

The outcomes of Experiment 03 are visually documented in Figure 5.3. Although the scrambled image in Figure 4.1 appears less scrambled compared to that in Figure 4.2, the experiment did not succeed in generating an abstract art image. Moreover, the reconstructed image failed to encapsulate any information from the original secret images. This observation leads to the conclusion that the Generators G1 and G2, as explained in Section 4.4.1.3, lack the capability to process such heavily distorted inputs and produce new abstract art. Consequently, future experiments will pivot towards employing milder forms of distortion that do not completely alter the original fused image information, unlike the approaches taken in experiments 02 and 03. This strategic shift aims to refine the generative model's ability to manipulate and reinterpret image data without losing its inherent value.

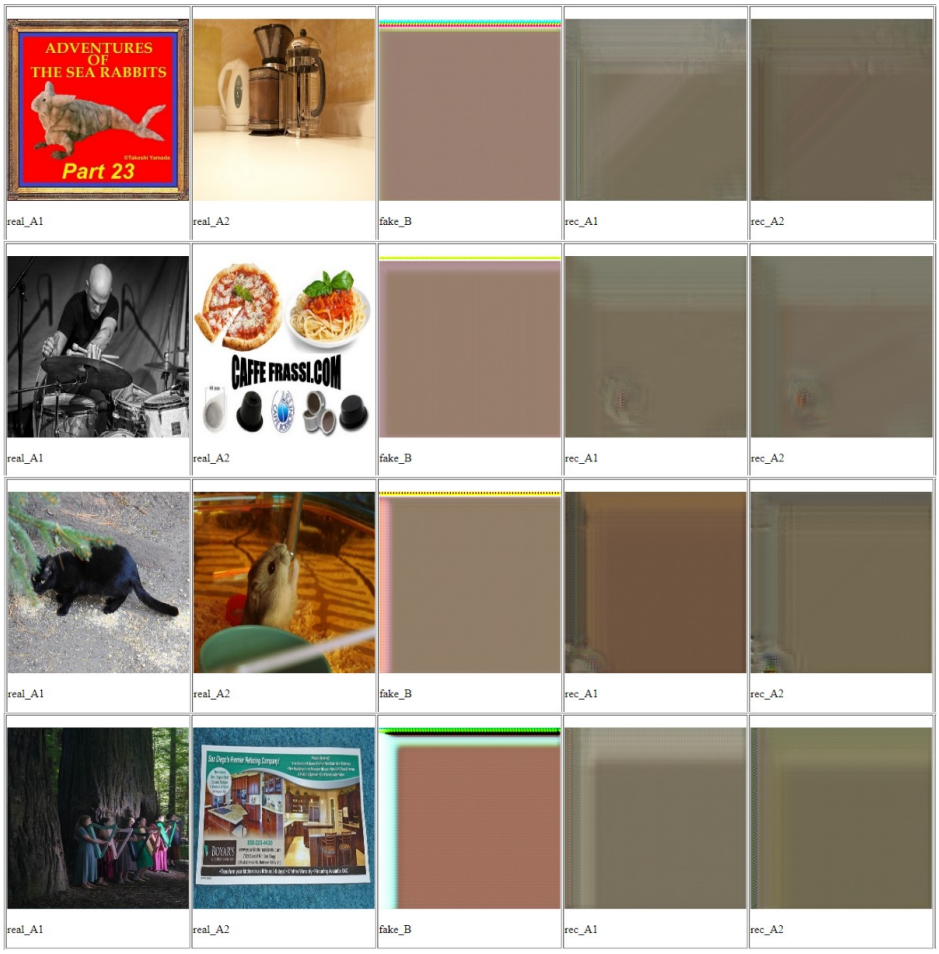


Figure 5.3: Visual results of experiment 03

5.1.4 Experiment 04

As explained in Section 3.6.4, we replicated the experimental setup detailed in Section 3.6.1, with the outcomes presented in Figure 5.4. Remarkably, completing the experiment required only 2 days, 10 hours, and 11 minutes to reach 500k iterations, signifying a substantial enhancement in efficiency compared to the initial experiments. Specifically, Experiment 01 necessitated approximately 1 day, 19 hours, and 12 minutes to reach just 183k iterations, illustrating the significant time reduction achieved in the latest trial.

Upon comparing the results with those from Experiment 01, as showcased in Figure 5.1, a notable improvement is observed. The generated stego images in this experiment were more effective at concealing the features of the secret images. Although the concealment wasn't absolute, the degree of obfuscation represents

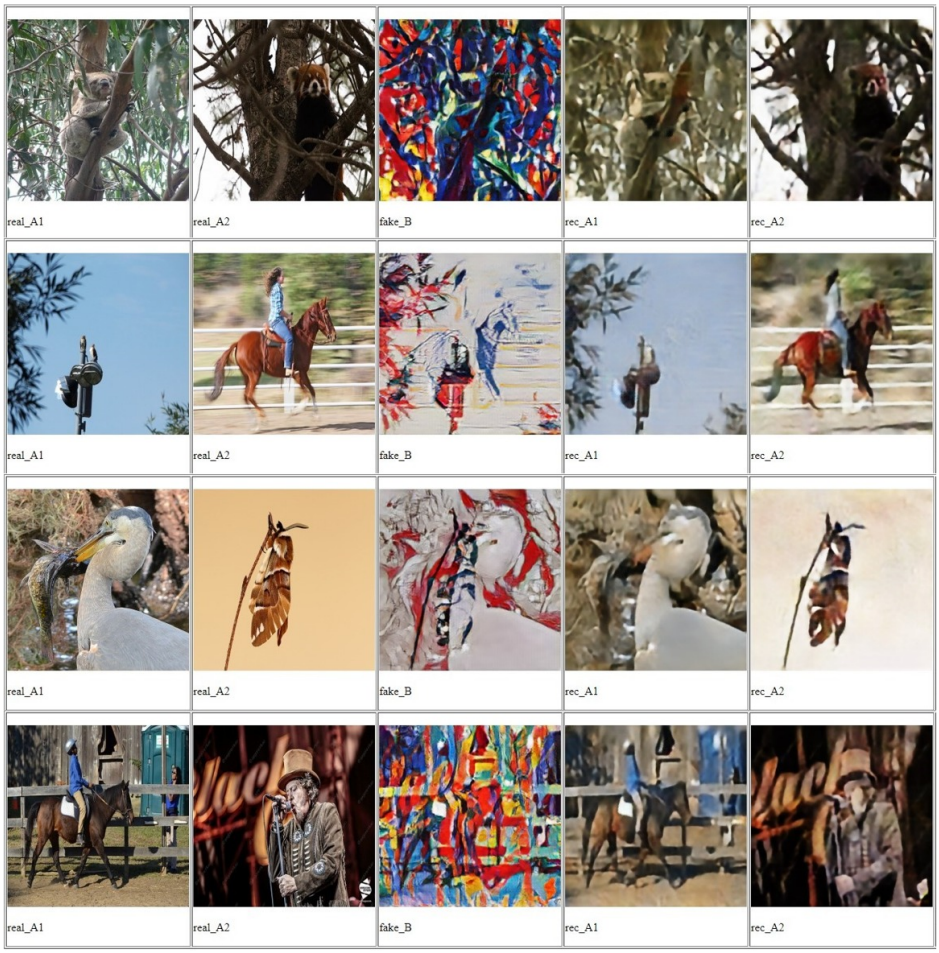


Figure 5.4: Visual results of experiment 04

a marked progression. Furthermore, a notable limitation in Experiment 01 was that only one of the recovered secret images retained colour information, with the other reverting to grayscale. This issue has been successfully addressed in the current experiment, as evidenced by both recovered secret images displaying colour information. This development not only enhances the visual quality of the recovered images but also signifies a critical advancement in our experimental methodology and outcome.

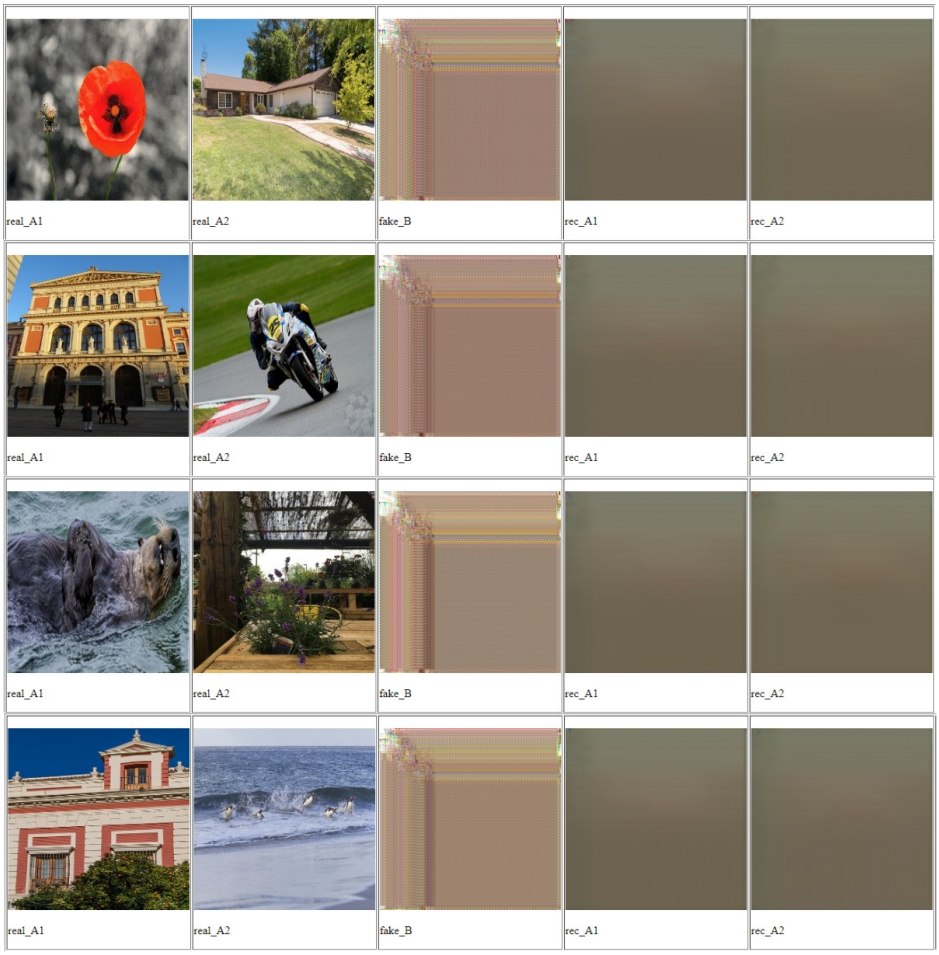


Figure 5.5: Visual results of experiment 05

5.1.5 Experiment 05

The results of Experiment 05 are depicted in Figure 5.5. Unfortunately, this experiment did not meet our success criteria; it failed to generate the intended abstract image. Additionally, it was unable to successfully recover any of the input secret images. This outcome underscores the need for further investigation and potential adjustments to our experimental approach to achieve the desired results in future iterations.

5.2 Evaluation

Given that Experiment 03 stands out as the most successful among the experiments conducted, it will be the focus of our in-depth evaluation. This evaluation will involve a comprehensive analysis sections 5.2.1, 5.2.2.

5.2.1 Image Quality Evaluation

The evaluation of the proposed model was conducted using a test dataset composed of 1000 images. The images generated by the model were evaluated to assess the quality of image reproduction. To provide a comprehensive evaluation of the reproduced images' performance, key metrics such as the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM) were calculated for each input image. These individual assessments were then averaged to yield a performance metric for the entire dataset, as detailed in Section 3.7.1. The findings from this thorough evaluation process are compiled and presented in Table 5.1, offering insights into the model's efficacy in image reproduction.

Evaluation	Averaged Value
RMSE	22.5
MSE	546.25
PSNR	21.39
SSIM	0.66

Table 5.1: Evaluation under RMSE, MSE, PSNR, SSIM.

The values obtained from this evaluation can be subjected to a comparison against existing image steganography methods. Table 5.2 shows the comparison of the RMSE, MSE, PSNR, and SSIM values between the multi image steganography model and 5.3 compared with other image steganography models widely utilized.

For the comparison in Table 5.2 we specifically isolated the RED channel in both the original and reconstructed secret images. This decision was informed by the methodology of Dharmawimala (2023), which exclusively utilized the RED channel in their steganographic process. Despite the absence of a scrambling component in our model, it presents a competitive challenge to the work of Dharmawimala (ibid.), particularly in terms of SSIM. SSIM is a crucial statistic metric as it provides a more perception-oriented evaluation of error, giving insight into the visual similarity between the original and stego images beyond mere pixel-based errors.

Evaluation	Dharmawimala (2023)	Proposed model(RED channel)
RMSE	8.74	20.02
MSE	77.87	435.84
PSNR	29.39	22.45
SSIM	0.71	0.69

Table 5.2: Comparison of RMSE, MSE, PSNR, and SSIM values between other multi image steganography models and the proposed model

Evaluation	Baluja (2017)	Q. Li et al. (2021)	X. Liu et al. (2020)	Proposed model
RMSE	-	-	-	22.50
MSE	-	76.1271	-	546.25
PSNR	27.51	29.23	36.23	21.39
SSIM	0.89	0.73	0.97	0.66

Table 5.3: Comparison of MSE, PSNR, and SSIM values between other image steganography models and the proposed model

Our proposed model, although not outperforming existing models in all quantitative metrics, brings forward innovative approaches that contribute to the field's understanding of GAN-based multi-image steganography. The model's SSIM, a measure of perceptual quality, is competitive and demonstrates the model's capacity to produce visually convincing stego images that are crucial for practical steganographic applications.

5.2.2 Hiding Capacity

As described in Section 3.7.3, the hiding capacity of coverless image steganography models refers to the number of bits a carrier image can represent in terms of bits per cover ($bits * cover^{-1}$). Table 5.4 presents the comparison of hiding capacity between steganography models and the proposed model.

Notably, the proposed model achieves a significant enhancement in both absolute and relative capacity metrics.

In absolute terms, the model exhibits the capability to embed a substantial 256x256x8x2x3 bits per image. This absolute capacity indicates the total number of bits that can be hidden within a stego image and is a critical metric for evaluating steganographic efficiency.

When evaluating the relative capacity, which is expressed as the number of bits per pixel that can be hidden, the proposed model demonstrates a remarkable

Method	Absolute capacity (bit/image)	Image size	Relative capacity (bit/pixel)
Hu et al. (2018)	300	64x64	0.073
Z. Zhang et al. (2019)	18.3-135.4	64x64	0.004-0.033
Zhou, Q. Wu, et al. (2017)	18	-	-
Cao et al. (2020)	896		-
Dharmawimala (2023)	256×256×8×2	256x256	16
X. Liu et al. (2020)	256×256×8×3	256x256	24
Proposed model	256×256×8×2x3	256x256	48

Table 5.4: Comparison of the hiding capacity

capacity of 48 bits per pixel. This is a marked improvement over other methods listed, including the advanced model by X. Liu et al. (2020), which shows a relative capacity of 24 bits per pixel.

The proposed model's capacity stands out, particularly in the context of its image size compatibility. It maintains this high embedding rate without a reduction in image size. This superior capacity suggests that the model could be highly effective for scenarios that demand the secure and efficient hiding of large quantities of secret images within an image.

Chapter 6

Conclusion

6.1 Conclusions about Research Questions

Given the main objective is to produce stego images embedding RGB colour secret images ensuring both a high degree of accuracy and concealment from the human eye, this dissertation explores the research question: "How can GAN-based coverless image steganography methods be extended to implement a model capable of multi-image steganography for RGB colour images"

To achieve this objective, a comprehensive review of the existing literature, as outlined in Chapter 2, was conducted. This review illuminated the foundational concepts underpinning the Cam-GAN model introduced by X. Liu et al. (2020), alongside contributions from Dharmawimala (2023), serving as critical reference points for this research. Drawing upon these key studies, the development of a novel model capable of executing multi-image steganography with RGB secret images was developed. The complexities of this development process, including the conceptualization and operational realization of the proposed model, are elaborated upon in Chapters 3 and 4.

This approach not only bridges the gap identified in the existing literature regarding the application of GAN-based techniques to RGB colour image steganography but also contributes a practical solution to the nuanced challenge of maintaining perceptual invisibility while ensuring data fidelity in the embedded images.

6.2 Conclusions about Research Problem

The research problem following this study was primarily focused on generating stego images with RGB colour secret images that maintain a satisfactory level of accuracy and concealment to the human eye. To achieve this, GANs were utilized,

and to provide additional security, a coverless image steganography approach was adopted.

This approach marks a significant advancement over existing methodologies, such as the multi-image steganography model proposed by Das (2021), which relies on cover modification steganography and thus offers lower security. It also challenges the Dharmawimala (2023) model, which excludes the usage of RGB secret images without even the scrambling component.

Compared to traditional image steganography methods, the proposed model not only achieves commendable results in maintaining concealment and accuracy but also demonstrates an improved embedding capacity. Thus, this study contributed to the domain of image steganography by exploring coverless multi-image steganography using GANs.

Furthermore, this model sets a groundwork for future research, particularly in exploring GANs within this context. It's also worth mentioning that the proposed model lays the groundwork for optimization and refinement, presenting a solid baseline from which subsequent models can be developed to enhance these metrics. Therefore, while the journey to refine these quantitative results continues, the proposed model stands as a testament to the potential and adaptability of GANs in the evolving domain of steganography.

6.3 Limitations

The scope of the study had to be shrunken to adhere to the timelines of the undertaken study and resource constraints. As a result, the proposed model was implemented to take two RGB images as input. This constraint prevents the model from being extended to accept more than two images or RGB images as input.

Furthermore, the dataset utilized for training was downsized, and the number of epochs in training was limited. It is reasonable to anticipate that extended training, with a larger dataset and a higher number of epochs, could have resulted in an increment in the model's accuracy in terms of image reproduction quality.

The time limitation of this study also limited our capacity to investigate the inclusion of new generator networks for generators G2 and G3, which may have better capabilities for image-to-image translation and recovery than the networks currently in use.

Furthermore, the complexity of the proposed model meant that exhaustive hyperparameter tuning was beyond the scope of this study. Such tuning has the potential to significantly refine model performance.

In light of these limitations, the current iteration of the model serves as a promising proof of concept. The study paves the way for subsequent research that could overcome these limitations, thus advancing the field of steganography

towards more robust, versatile, and higher-performing solutions.

6.4 Implications for further research

The exploration of advanced image-to-image translation models, such as pix2pix by Isola et al. (2016), offers a promising avenue for enhancing the capabilities of current systems. These models, which enable the better translation of fused images to more realistic abstract art images or any type of stego image, represent a significant leap forward from existing technologies. Integrating advanced image-to-image translation models into the proposed system holds promise for refining its performance. The main problem is the model's ability to accurately reproduce the original, fused image from the generated stego image. This capability is not typically found in current state-of-the-art image-to-image translation models, which often focus on the translation aspect without the necessity of reconstructing the original images.

To address this gap, future research could explore developing or enhancing translation models with a dual focus: ensuring high-fidelity translations while also maintaining the ability to reverse the process. This would involve not only the accurate rendering of visual content in the stego image but also the preservation of all necessary information to recover the original, fused images without loss.

In addition to model architecture improvements, there is significant potential in optimizing the model's hyperparameters. Rigorous hyperparameter tuning has the potential to fine-tune the model's learning process, likely resulting in a notable increase in steganographic performance.

Incorporating transform domain approaches, as suggested by Dharmawimala (2023), could facilitate the merging of multiple secret images, a process that may prove more efficient than current spatial domain methods. This shift could lead to improvements in the fusion of images and enhance the overall quality of the merged output.

The utilization of sophisticated feature extraction models like VGG16, as introduced by (Simonyan and Zisserman 2014), could also be instrumental in extracting intricate content details from secret images. By encoding these details into the generated image, one could significantly improve the robustness and fidelity of image reproduction.

Bibliography

Baluja, Shumeet (Dec. 2017). "Hiding images in plain sight: deep steganography". In: vol. 30, pp. 2066–2076. URL: https://papers.nips.cc/paper/6802-hiding-images-in-plain-sight-deep-steganography.pdf.

Boehm, Benedikt (Oct. 2014). *StegExpose - A Tool for Detecting LSB Steganography*. URL: https://arxiv.org/abs/1410.6656.

Cao, Yi et al. (Dec. 2020). "Coverless information hiding based on the generation of anime characters". In: *Eurasip Journal on Image and Video Processing* 2020.1. DOI: 10.1186/s13640-020-00524-4. URL: https://doi.org/10.1186/s13640-020-00524-4.

Cheddad, Abbas et al. (Mar. 2010). "Digital image steganography: Survey and analysis of current methods". In: *Signal Processing* 90.3, pp. 727–752. DOI: 10.1016/j.sigpro.2009.08.010. URL: https://doi.org/10.1016/j.sigpro.2009.08.010.

Chu, Casey, Andrey Zhmoginov, and Mark Sandler (Dec. 8, 2017). "CycleGAN, a master of steganography". In: *arXiv (Cornell University)*. URL: http://export.arxiv.org/pdf/1712.02950.

Cui, Qi N. et al. (Apr. 2019). "Image Steganography Based on Foreground Object Generation by Generative Adversarial Networks in Mobile Edge Computing With Internet of Things". In: *IEEE Access* 7, pp. 90815–90824. DOI: 10.1109/access.2019.2913895. URL: https://doi.org/10.1109/access.2019.2913895.

Das, Abhishek (Jan. 2021). *Multi-Image Steganography Using Deep Neural Networks*. URL: https://arxiv.org/abs/2101.00350.

Dharmawimala, Yashithi (May 2023). "Coverless Multi-Image Steganography using Generative Adversarial Networks". In: p. 123.

Duan, Xintao (Feb. 2018). *Coverless information hiding based on Generative Model*. URL: https://arxiv.org/abs/1802.03528.

Fridrich, Jessica (Aug. 2010). "Steganography in digital media: principles, algorithms, and applications". In: *Choice Reviews Online* 47.12, pp. 47–6907. DOI:

10.5860/choice.47-6907. URL: https://doi.org/10.5860/choice.47-6907.

Fu, Zhangjie, Fan Wang, and Xu Cheng (Dec. 2020). "The secure steganography for hiding images via GAN". In: *EURASIP Journal on Image and Video Processing* 2020.1. DOI: 10.1186/s13640-020-00534-2. URL: https://doi.org/10.1186/s13640-020-00534-2.

Gontier, Camille, Jakob Jordan, and Mihai A. Petrovici (Jan. 28, 2022). "DELAUNAY: a dataset of abstract art for psychophysical and machine learning research". In: *arXiv (Cornell University)*. DOI: 10.48550/arxiv.2201.12123. URL: http://arxiv.org/abs/2201.12123.

Goodfellow, Ian et al. (Oct. 15, 2017). "GAN(Generative Adversarial Nets)". In: *Journal of Japan Society for Fuzzy Theory and Intelligent Informatics* 29.5, p. 177. DOI: 10.3156/jsoft.29.5_177_2. URL: https://doi.org/10.3156/jsoft.29.5_177_2.

Hu, Donghui et al. (July 2018). "A Novel Image Steganography Method via Deep Convolutional Generative Adversarial Networks". In: *IEEE Access* 6, pp. 38303–38314. DOI: 10.1109/access.2018.2852771. URL: https://doi.org/10.1109/access.2018.2852771.

Isola, Phillip et al. (Nov. 21, 2016). "Image-to-Image Translation with Conditional Adversarial Networks". In: *arXiv (Cornell University)*. URL: http://export.arxiv.org/pdf/1611.07004.

Johnson and Jajodia (Jan. 1998). "Exploring steganography: Seeing the unseen". In: *IEEE Computer* 31.2, pp. 26–34. DOI: 10.1109/mc.1998.4655281. URL: https://doi.org/10.1109/mc.1998.4655281.

Karras, Tero, Samuli Laine, and Timo Aila (Dec. 2018). "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *arXiv (Cornell University)*. DOI: 10.48550/arxiv.1812.04948. URL: http://arxiv.org/abs/1812.04948.

Li, Min, Ting Liang, and Yujie He (Jan. 1, 2013). "Arnold Transform based image scrambling Method". In: *Advances in intelligent systems research/Advances in Intelligent Systems Research*. DOI: 10.2991/icmt-13.2013.160. URL: https://doi.org/10.2991/icmt-13.2013.160.

Li, Qi et al. (July 2021). "CCCIH: Content-consistency Coverless Information Hiding Method Based on Generative Models". In: *Neural Processing Letters* 53.6, pp. 4037–4046. DOI: 10.1007/s11063-021-10582-y. URL: https://doi.org/10.1007/s11063-021-10582-y.

Liu, Jia et al. (Mar. 2020). "Recent Advances of Image Steganography With Generative Adversarial Networks". In: *IEEE Access* 8, pp. 60575–60597. DOI: 10.1109/access.2020.2983175. URL: https://doi.org/10.1109/access.2020.2983175.

Liu, Mingming et al. (Dec. 18, 2017). "Coverless Information Hiding Based on Generative adversarial networks." In: *arXiv (Cornell University)*. URL: https://arxiv.org/pdf/1712.06951.pdf.

Liu, Xiyao et al. (Oct. 2020). "Camouflage Generative Adversarial Network: Coverless Full-image-to-image Hiding". In: DOI: 10.1109/smc42975.2020.9283054. URL: https://doi.org/10.1109/smc42975.2020.9283054.

Mielikainen, Jarno (Apr. 2006). "LSB matching revisited". In: *IEEE Signal Processing Letters* 13.5, pp. 285–287. DOI: 10.1109/lsp.2006.870357. URL: https://doi.org/10.1109/lsp.2006.870357.

Perera, Pramuditha, Mahdi Abavisani, and Vishal M. Patel (Nov. 25, 2017). "IN2I : Unsupervised Multi-Image-to-Image Translation using Generative Adversarial Networks". In: *arXiv (Cornell University)*. DOI: 10.48550/arxiv.1711.09334. URL: http://arxiv.org/abs/1711.09334.

Qian, Yinlong et al. (Mar. 2015). "Deep learning for steganalysis via convolutional neural networks". In: SPIE. DOI: 10.1117/12.2083479. URL: https://doi.org/10.1117/12.2083479.

Ruohan, Meng et al. (June 2019). "A Steganography Algorithm Based on CycleGAN for Covert Communication in the Internet of Things". In: *IEEE Access* 7, pp. 90574–90584. DOI: 10.1109/access.2019.2920956. URL: https://doi.org/10.1109/access.2019.2920956.

Russakovsky, Olga et al. (Apr. 11, 2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y.

Shi, Haichao (July 2017). *SSGAN: Secure Steganography Based on Generative Adversarial Networks*. URL: https://arxiv.org/abs/1707.01613.

Simmons, Gustavus J. (Jan. 1984). *The Prisoners' Problem and the Subliminal Channel*, pp. 51–67. DOI: 10.1007/978-1-4684-4730-9\{_}5. URL: https://doi.org/10.1007/978-1-4684-4730-9_5.

Simonyan, Karen and Andrew Zisserman (Jan. 1, 2014). "Very deep convolutional networks for Large-Scale image recognition". In: *arXiv (Cornell University)*. DOI: 10.48550/arxiv.1409.1556. URL: https://arxiv.org/abs/1409.1556.

Volkhonskiy, Denis (Mar. 2017). *Steganographic Generative Adversarial Networks*. URL: https://arxiv.org/abs/1703.05502.

Wang, Yaofei et al. (Jan. 2020). "Non-Additive Cost Functions for Color Image Steganography Based on Inter-Channel Correlations and Differences". In: *IEEE Transactions on Information Forensics and Security* 15, pp. 2081–2095. DOI: 10.1109/tifs.2019.2956590. URL: https://doi.org/10.1109/tifs.2019.2956590.

Wei, Li-Yi (Oct. 1999). "Deterministic texture analysis and synthesis using tree structure vector quantization". In: DOI: 10.1109/sibgra.1999.805726. URL: https://doi.org/10.1109/sibgra.1999.805726.

Wu, Hao et al. (Oct. 2005). "Image steganographic scheme based on pixel-value differencing and LSB replacement methods". In: *IEE Proceedings - Vision, Image and Signal Processing* 152.5, p. 611. DOI: 10.1049/ip-vis:20059022. URL: https://doi.org/10.1049/ip-vis:20059022.

Xu, Guanshuo, Hanzhou Wu, and Yun Q. Shi (Mar. 2016). "Structural Design of Convolutional Neural Networks for Steganalysis". In: *IEEE Signal Processing Letters* 23.5, pp. 708–712. DOI: 10.1109/lsp.2016.2548421. URL: https://doi.org/10.1109/lsp.2016.2548421.

Ye, Jian, Jiangqun Ni, and Yang Yi (June 2017). "Deep Learning Hierarchical Representations for Image Steganalysis". In: *IEEE Transactions on Information Forensics and Security* 12.11, pp. 2545–2557. DOI: 10.1109/tifs.2017.2710946. URL: https://doi.org/10.1109/tifs.2017.2710946.

Zhang, Kevin Alex (Jan. 2019). *SteganoGAN: High Capacity Image Steganography with GANs*. URL: https://arxiv.org/abs/1901.03892.

Zhang, Zhuo et al. (Sept. 11, 2019). "Generative reversible data hiding by Image-to-Image translation via GANS". In: *Security and Communication Networks* 2019, pp. 1–10. DOI: 10.1155/2019/4932782. URL: https://doi.org/10.1155/2019/4932782.

Zheng, Ziqiang (May 2019). *EncryptGAN: Image Steganography with Domain Transform*. URL: https://arxiv.org/abs/1905.11582.

Zhou, Zhili, Huiyu Sun, et al. (Aug. 2015). *Coverless Image Steganography Without Embedding*. Springer Science+Business Media, pp. 123–132. DOI: 10.1007/978-3-319-27051-7\{_}11. URL: https://doi.org/10.1007/978-3-319-27051-7_11.

Zhou, Zhili, Qinghua Wu, et al. (Sept. 2017). "Coverless Image Steganography Using Histograms of Oriented Gradients-Based Hashing Algorithm". In: *Journal of Internet Technology* 18.5, pp. 1177–1184. DOI: 10.6138/jit.2017.18.5.20160815b. URL: https://jit.ndhu.edu.tw/article/view/1546.

Appendix A

Code Listings

```
from scipy import ndimage
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import cv2
import os

datasetA = '/hdd/2019CS077/Dataset/trainA1/*.JPEG'
datasetB = '/hdd/2019CS077/Dataset/trainA2/*.JPEG'
destination = '/hdd/2019CS077/Code/v0/datasets/maps/trainA/'

dataA = sorted(glob.glob(datasetA))
dataB = sorted(glob.glob(datasetB))

count = 0

for i, j in zip(dataA, dataB):

    # Limiting the dataset to only 5000 images
    count += 1
    if (count > 5000):
        break

    img_A = cv2.imread(i)
    img_B = cv2.imread(j)

    # Resizing the image and turning to grayscale
    dim = (256, 256)
    img_A = cv2.resize(img_A, dim, interpolation=cv2.INTER_LINEAR)
    img_B = cv2.resize(img_B, dim, interpolation=cv2.INTER_LINEAR)

    # Vertically stack images
    stacked_image = cv2.vconcat([img_A, img_B])

    # Save the stacked image
    # Use input image's filename for the output
    output_filename = os.path.basename(i)
    output_path = os.path.join(destination + str(count) + '.jpeg')
    cv2.imwrite(output_path, stacked_image)
print('%d images generated !' % (count-1))
```

Figure A.1: Train data prepossessing

```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import cv2
import os

datasetA = '/hdd/2019CS077/Dataset/testA1/*.JPEG'
datasetB = '/hdd/2019CS077/Dataset/testA2/*.JPEG'
destination = '/hdd/2019CS077/TestCode/v0/datasets/maps/testA/'

dataA = sorted(glob.glob(datasetA))
dataB = sorted(glob.glob(datasetB))

count = 0

for i, j in zip(dataA, dataB):

    count += 1
    if (count > 1000):
        break

    img_A = cv2.imread(i)
    img_B = cv2.imread(j)

    dim = (256, 256)
    img_A = cv2.resize(img_A, dim, interpolation=cv2.INTER_AREA)
    img_B = cv2.resize(img_B, dim, interpolation=cv2.INTER_AREA)

    line = img_A

    # Vertically stack images
    stacked_image = cv2.vconcat([img_A, img_B])

    # Save the stacked image
    # Use input image's filename for the output
    output_filename = os.path.basename(i)
    output_path = os.path.join(destination + str(count) + '.jpeg')
    cv2.imwrite(output_path, stacked_image)
print('%d images generated !' % (count))
```

Figure A.2: Test data prepossessing

```
class ResnetGeneratorMM(nn.Module):
    def __init__(self, input_nc=1, output_nc=3, ngf=64, norm_layer=nn.BatchNorm2d, use_dropout=False, n_blocks=6, gpu_ids=[], padding_type='reflect'):
        assert (n_blocks >= 0)
        super(ResnetGeneratorMM, self).__init__()
        self.input_nc = input_nc
        self.output_nc = 3
        self.ngf = ngf
        self.gpu_ids = gpu_ids
        if type(norm_layer) == functools.partial:
            use_bias = norm_layer.func == nn.InstanceNorm2d
        else:
            use_bias = norm_layer == nn.InstanceNorm2d
        model1 = [nn.ReflectionPad2d(3),

                  nn.Conv2d(input_nc, ngf, kernel_size=7, padding=0,
                            bias=use_bias),
                  norm_layer(ngf),
                  nn.ReLU(True)]

        model2 = [nn.ReflectionPad2d(3),
                  nn.Conv2d(input_nc, ngf, kernel_size=7, padding=0,
                            bias=use_bias),
                  norm_layer(ngf),
                  nn.ReLU(True)]

        n_downsampling = 2
        for i in range(n_downsampling):
            mult = 2**i
            model1 += [nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size=3,
                                 stride=2, padding=1, bias=use_bias),
                       norm_layer(ngf * mult * 2),
                       nn.ReLU(True)]

            model2 += [nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size=3,
                                 stride=2, padding=1, bias=use_bias),
                       norm_layer(ngf * mult * 2),
                       nn.ReLU(True)]

        pre_f_blocks = 4
        pre_l_blocks = 7
        mult = 2**n_downsampling

        for i in range(pre_f_blocks):
            model1 += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                   norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]
            model2 += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                   norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

        model_pre = []
        model_post = []
```

Figure A.3: Encoder network part 01

```
    for i in range(pre_f_blocks, pre_l_blocks):
        model_pre += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                  norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

    for i in range(pre_l_blocks, n_blocks):
        model_post += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                   norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

    model_fusion = []

    model_fusion = [nn.Conv2d(ngf * mult * 2, ngf * mult, kernel_size=3, padding=1, bias=use_bias),
                    norm_layer(ngf * mult),
                    nn.ReLU(True)]

    model = []
    for i in range(n_downsampling):
        mult = 2**(n_downsampling - i)
        model_post += [nn.ConvTranspose2d(ngf * mult, int(ngf * mult / 2),
                                          kernel_size=3, stride=2,
                                          padding=1, output_padding=1,
                                          bias=use_bias),
                       norm_layer(int(ngf * mult / 2)),
                       nn.ReLU(True)]
    model_post += [nn.ReflectionPad2d(3)]
    model_post += [nn.Conv2d(ngf, output_nc, kernel_size=7, padding=0)]
    model_post += [nn.Tanh()]
    input_nc2 = 3
    model = [nn.ReflectionPad2d(3),
             nn.Conv2d(input_nc2, ngf, kernel_size=7,
                       padding=0, bias=use_bias),
             norm_layer(ngf),
             nn.ReLU(True)]

    n_downsampling = 2
    for i in range(n_downsampling):  # add downsampling layers
        mult = 2 ** i
        model += [nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size=3, stride=2, padding=1, bias=use_bias),
                  norm_layer(ngf * mult * 2),
                  nn.ReLU(True)]

    mult = 2 ** n_downsampling
    for i in range(n_blocks):       # add ResNet blocks

        model += [ResnetBlock(ngf * mult, padding_type=padding_type,
                              norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]
```

```
    for i in range(n_downsampling):  # add upsampling layers
        mult = 2 ** (n_downsampling - i)
        model += [nn.ConvTranspose2d(ngf * mult, int(ngf * mult / 2),
                                     kernel_size=3, stride=2,
                                     padding=1, output_padding=1,
                                     bias=use_bias),
                  norm_layer(int(ngf * mult / 2)),
                  nn.ReLU(True)]
    model += [nn.ReflectionPad2d(3)]
    model += [nn.Conv2d(ngf, output_nc, kernel_size=7, padding=0)]
    model += [nn.Tanh()]

    self.model = nn.Sequential(*model)
    self.model1 = nn.Sequential(*model1)
    self.model2 = nn.Sequential(*model2)
    self.model_pre = nn.Sequential(*model_pre)
    self.model_post = nn.Sequential(*model_post)
    self.model_fusion = nn.Sequential(*model_fusion)

def forward(self, input, input2):
    m1 = self.model1(input)
    m2 = self.model2(input2)
    latent = self.model_pre(self.model_fusion(torch.cat([m1, m2], dim=1)))
    intermediate = self.model_post(latent)
    out = self.model(intermediate)
    return out, latent, intermediate
```

Figure A.4: Encoder network part 02

```
class ResnetGeneratorMMReverse(nn.Module):
    def __init__(self, input_nc, output_nc, ngf=64, norm_layer=nn.BatchNorm2d, use_dropout=False, n_blocks=6, gpu_ids=[], padding_type='reflect'):
        assert (n_blocks >= 0)
        super(ResnetGeneratorMMReverse, self).__init__()
        self.input_nc = input_nc
        self.output_nc = output_nc
        self.ngf = ngf
        self.gpu_ids = gpu_ids
        if type(norm_layer) == functools.partial:
            use_bias = norm_layer.func == nn.InstanceNorm2d
        else:
            use_bias = norm_layer == nn.InstanceNorm2d

        model_pre1 = [nn.ReflectionPad2d(3),
                      nn.Conv2d(3, ngf, kernel_size=7,
                                padding=0, bias=use_bias),
                      norm_layer(ngf),
                      nn.ReLU(True)]

        n_downsampling = 2
        for i in range(n_downsampling):  # add downsampling layers
            mult = 2 ** i
            model_pre1 += [nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size=3, stride=2, padding=1, bias=use_bias),
                           norm_layer(ngf * mult * 2),
                           nn.ReLU(True)]

        mult = 2 ** n_downsampling
        for i in range(n_blocks):       # add ResNet blocks

            model_pre1 += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                       norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

        for i in range(n_downsampling):  # add upsampling layers
            mult = 2 ** (n_downsampling - i)
            model_pre1 += [nn.ConvTranspose2d(ngf * mult, int(ngf * mult / 2),
                                              kernel_size=3, stride=2,
                                              padding=1, output_padding=1,
                                              bias=use_bias),
                           norm_layer(int(ngf * mult / 2)),
                           nn.ReLU(True)]
        model_pre1 += [nn.ReflectionPad2d(3)]
        model_pre1 += [nn.Conv2d(ngf, 3, kernel_size=7, padding=0)]
        model_pre1 += [nn.Tanh()]

        self.model_pre1 = nn.Sequential(*model_pre1)

        model = [nn.ReflectionPad2d(3),

                 nn.Conv2d(input_nc, ngf, kernel_size=7, padding=0,
                           bias=use_bias),
                 norm_layer(ngf),
                 nn.ReLU(True)]
```

Figure A.5: Decoder network part 01

```
        n_downsampling = 2
        for i in range(n_downsampling):
            mult = 2**i
            model += [nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size=3,
                                stride=2, padding=1, bias=use_bias),
                      norm_layer(ngf * mult * 2),
                      nn.ReLU(True)]

        pre_f_blocks = 4
        pre_l_blocks = 7
        mult = 2**n_downsampling

        for i in range(pre_f_blocks):
            model += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                  norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

        model_pre = []
        for i in range(pre_f_blocks, pre_l_blocks):
            model_pre += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                      norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

        model_post1 = []
        model_post2 = []
        for i in range(pre_l_blocks, n_blocks):
            model_post1 += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                        norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]
            model_post2 += [ResnetBlock(ngf * mult, padding_type=padding_type,
                                        norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

        for i in range(n_downsampling):
            mult = 2**(n_downsampling - i)
            model_post1 += [nn.ConvTranspose2d(ngf * mult, int(ngf * mult / 2),
                                               kernel_size=3, stride=2,
                                               padding=1, output_padding=1,
                                               bias=use_bias),
                            norm_layer(int(ngf * mult / 2)),
                            nn.ReLU(True)]

            model_post2 += [nn.ConvTranspose2d(ngf * mult, int(ngf * mult / 2),
                                               kernel_size=3, stride=2,
                                               padding=1, output_padding=1,
                                               bias=use_bias),
                            norm_layer(int(ngf * mult / 2)),
                            nn.ReLU(True)]

        model_post1 += [nn.ReflectionPad2d(3)]
        model_post1 += [nn.Conv2d(ngf, output_nc, kernel_size=7, padding=0)]
        model_post1 += [nn.Tanh()]

        model_post2 += [nn.ReflectionPad2d(3)]
        model_post2 += [nn.Conv2d(ngf, output_nc, kernel_size=7, padding=0)]
        model_post2 += [nn.Tanh()]

        self.model_post1 = nn.Sequential(*model_post1)
        self.model_post2 = nn.Sequential(*model_post2)
        self.model_pre = nn.Sequential(*model_pre)
        self.model = nn.Sequential(*model)

    def forward(self, input):
        # noise_tensor = torch.randn(input.size()) * 0.001
        # noise_tensor=noise_tensor.to('cuda')
        # add noise to the image tensor
        # noisy_tensor = input + noise_tensor

        # clamp the tensor to be between 0 and 1
        # noisy_tensor = torch.clamp(noisy_tensor, min=0.0, max=1.0)

        intermediate = self.model_pre1(input)
        latent = self.model(intermediate)
        fuse_ip = self.model_pre(latent)
        out1 = self.model_post1(fuse_ip)
        out2 = self.model_post2(fuse_ip)
        return out1, out2, latent, intermediate
```

Figure A.6: Decoder network part 02

```python
class NLayerDiscriminator(nn.Module):
    def __init__(self, input_nc, ndf=64, n_layers=3, norm_layer=nn.BatchNorm2d, use_sigmoid=False, gpu_ids=[]):
        super(NLayerDiscriminator, self).__init__()
        self.gpu_ids = gpu_ids
        if type(norm_layer) == functools.partial:
            use_bias = norm_layer.func == nn.InstanceNorm2d
        else:
            use_bias = norm_layer == nn.InstanceNorm2d

        kw = 4
        padw = 1
        sequence = [
            nn.Conv2d(input_nc, ndf, kernel_size=kw, stride=2, padding=padw),
            nn.LeakyReLU(0.2, True)      Mahe077, 8 months ago • Initial commit
        ]

        nf_mult = 1
        nf_mult_prev = 1
        for n in range(1, n_layers):
            nf_mult_prev = nf_mult
            nf_mult = min(2**n, 8)
            sequence += [
                nn.Conv2d(ndf * nf_mult_prev, ndf * nf_mult,
                          kernel_size=kw, stride=2, padding=padw, bias=use_bias),
                norm_layer(ndf * nf_mult),
                nn.LeakyReLU(0.2, True)
            ]

        nf_mult_prev = nf_mult
        nf_mult = min(2**n_layers, 8)
        sequence += [
            nn.Conv2d(ndf * nf_mult_prev, ndf * nf_mult,
                      kernel_size=kw, stride=1, padding=padw, bias=use_bias),
            norm_layer(ndf * nf_mult),
            nn.LeakyReLU(0.2, True)
        ]

        sequence += [nn.Conv2d(ndf * nf_mult, 1,
                               kernel_size=kw, stride=1, padding=padw)]

        if use_sigmoid:
            sequence += [nn.Sigmoid()]

        self.model = nn.Sequential(*sequence)

    def forward(self, input):
        if len(self.gpu_ids) and isinstance(input.data, torch.cuda.FloatTensor):
            return nn.parallel.data_parallel(self.model, input, self.gpu_ids)
        else:
            return self.model(input)
```

Figure A.7: PatchGAN network

```python
class Arnold:

    def __init__(self, a:int, b:int, rounds:int):
        # Parameters
        self.__a = a
        self.__b = b
        self.__rounds = rounds

    def mapping(self, s:np.shape):
        x, y = np.meshgrid(range(s[0]), range(s[0]), indexing="ij")
        xmap = (self.__a*self.__b*x + x + self.__a*y) % s[0]
        ymap = (self.__b*x + y) % s[0]
        return xmap, ymap

    def inverseMapping(self, s:np.shape):
        x, y = np.meshgrid(range(s[0]), range(s[0]), indexing="ij")
        xmap = (x - self.__a*y) % s[0]
        ymap = (-self.__b*x + self.__a*self.__b*y + y) % s[0]
        return xmap, ymap

    def applyTransformTo(self, image:np.ndarray):
        xm, ym = self.mapping(image.shape)
        img = image
        for r in range(self.__rounds):
            img = img[xm, ym]
        return img

    def applyInverseTransformTo(self, image:np.ndarray):
        xm, ym = self.inverseMapping(image.shape)
        img = image
        for r in range(self.__rounds):
            img = img[xm, ym]
        return img
```

Figure A.8: Improved Arnold Transformation used in experiment 3.6.2

```python
class ArnoldTransformation_v1():
    def scramble(self, image, iterations=1):
        _, channels, height, width = image.shape
        scramble_image = image.clone()

        for _ in range(iterations):
            for x in range(width):
                for y in range(height):
                    x_prime = (x + y) % width
                    y_prime = (x + 2 * y) % height
                    scramble_image[0, :, y_prime,
                                   x_prime] = image[0, :, y, x]

            image = scramble_image.clone()

        return scramble_image

    def reverseScramble(self, image, iterations=1):
        _, channels, height, width = image.shape
        scramble_image = image.clone()

        for _ in range(iterations):
            for x in range(width):
                for y in range(height):
                    x_prime = (2 * x - y) % width
                    y_prime = (y - x) % height
                    scramble_image[0, :, y_prime,
                                   x_prime] = image[0, :, y, x]

            image = scramble_image.clone()

        return scramble_image
```

Figure A.9: Arnold Transformation used in experiment 3.6.3

```python
def shuffle_image_blocks(tensor, opt):
    # Check if GPU is available and move tensor to GPU
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    tensor = tensor.to(device)

    # Check divisibility
    B, C, H, W = tensor.shape

    shuffled_tensor = torch.zeros_like(tensor)

    for c in range(C):
        # Divide into blocks and shuffle
        blocks = tensor[0, c].unfold(
            0, opt.bs, opt.bs).unfold(1, opt.bs, opt.bs)
        blocks = blocks.contiguous().view(-1, opt.bs, opt.bs)
        shuffled_blocks = torch.zeros_like(blocks)
        shuffled_blocks[opt.index] = blocks[opt.perm]

        # Reassemble the image
        for i in range(opt.num_blocks):
            for j in range(opt.num_blocks):
                idx = i * opt.num_blocks + j
                shuffled_tensor[0, c, i*opt.bs:(i+1)*opt.bs,
                                j*opt.bs:(j+1)*opt.bs] = shuffled_blocks[idx]
    return shuffled_tensor

def reverse_shuffle_image_blocks(tensor, opt):
    # Check if GPU is available and move tensor to GPU
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    tensor = tensor.to(device)

    # Check divisibility
    B, C, H, W = tensor.shape

    shuffled_tensor = torch.zeros_like(tensor)

    for c in range(C):
        # Divide into blocks and shuffle
        blocks = tensor[0, c].unfold(
            0, opt.bs, opt.bs).unfold(1, opt.bs, opt.bs)
        blocks = blocks.contiguous().view(-1, opt.bs, opt.bs)
        shuffled_blocks = torch.zeros_like(blocks)
        shuffled_blocks[opt.perm] = blocks[opt.index]

        # Reassemble the image
        for i in range(opt.num_blocks):
            for j in range(opt.num_blocks):
                idx = i * opt.num_blocks + j
                shuffled_tensor[0, c, i*opt.bs:(i+1)*opt.bs,
                                j*opt.bs:(j+1)*opt.bs] = shuffled_blocks[idx]
    return shuffled_tensor
```

Figure A.10: Shuffling Algorithm used in experiment 3.6.5