IMPROVED IMAGE STEGANOGRAPHY SYSTEM USING CONVOLUTIONAL NEURAL NETWORKS

BY

OFOEGBU, CHRISTOPHER IFEANYI (B.TECH, IFT FUTO)

20194204068

A THESIS SUBMITTED TO THE POST GRADUATE SCHOOL FEDERAL UNIVERSITY OF TECHNOLOGY OWERRI

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE (M.Sc.) DEGREE IN COMPUTER SCIENCE

NOVEMBER, 2024.

CERTIFICATION

This is to certify that this thesis **"Improved Image Steganography System Using Convolutional Neural Networks'** was carried out by **Ofoegbu, Christopher Ifeanyi (20194204068)** in partial fulfilment for the award of the degree of M.Sc. in Computer Science in the Department Computer Science of the Federal University of Technology, Owerri.

Dr. (Mrs.) E. C. Nwokorie
Supervisor

17/12/2024
Date

Dr. (Mrs.) U. C. Onyemauche
Co-Supervisor

17/12/2024
Date

Dr. (Mrs.) J. N. Odii
Head of Department

17/12/2024
Date

Prof. (Mrs.) U. F. Eze
Dean, SICT

18/12/24
Date

Prof. (Mrs.) J. N. Nwosu
Dean, PGS

Date

Prof. Moses Okechukwu Onyesolu
External Examiner

Date

DEDICATION

This thesis is dedicated to my father, Engr. E.O. Ofoegbu and my mother Dr. (Mrs.) C. G. Ofoegbu.

ACKNOWLEDGEMENTS

I extend my profound gratitude to the Almighty God for His unmerited grace and unwavering support throughout the duration of this program.

My sincere appreciation goes to my esteemed supervisors, Dr. (Mrs.) E.C. Nwokorie and Dr. (Mrs.) U.C. Onyemauche whose invaluable guidance and expertise were instrumental in shaping this work. I am equally indebted to Dr. S.A. Okolie, the former PG coordinator, for his concise yet pertinent corrections and for facilitating the entire process. I am deeply grateful to Dr. (Mrs.) J.N. Odii, the Head of Department, for her encouragement and insightful contributions during the course of this work. Special acknowledgment is also due to Dr. C.O. Ikerionwu, the reviewer of this work, for his meticulous efforts in reviewing and enforcing corrections that elevated the work to acceptable standards.

I express my gratitude to Prof. (Mrs.) U. F. Eze, the Dean (SICT) for her lenient responses whenever approached, and to Dr. I.I. Ayogu, the PG coordinator, for his valuable inputs and corrections throughout this endeavor. I am deeply indebted to all my lecturers, particularly Dr. (Mrs.) U.C. Onyemauche, Dr. D.O. Njoku, Dr. (Mrs.) M.E. Benson Emenike, Dr. (Mrs.) C.I. Okpalla, and all other lecturers in the Department of Computer Science and the School of SICT, whose scrutiny and contributions greatly contributed to the success of this thesis.

Special appreciation is reserved for my dear mother, Dr. (Mrs.) C.G. Ofoegbu, my father, Engr. E.O. Ofoegbu, my wife, Stephanie Ofoegbu, my sister, Anita Ofoegbu, and my brother, Chigozie Ofoegbu, for their unwavering encouragements and supports.

I am grateful to my colleagues, Amadi Christian and Ugonna Okorie, for their spirit of teamwork and collaboration.

Finally, I acknowledge the support of all friends and well-wishers whose encouragement played a significant role in the completion of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This study presents an innovative image steganography method that combines convolutional autoencoders with the Residual Network (ResNet) architecture to address the limitations of traditional techniques. The method aims to securely embed data within images while maintaining high imperceptibility and efficiency. Existing methods struggle with balancing image quality, embedding capacity, and computational complexity, highlighting the need for advanced solutions. This approach enables the concealment of color images within others, ensuring secure data transmission and reducing computational complexities. The method utilizes ResNet-based preprocessing for feature extraction and embedding operations, evaluated using the CIFAR dataset (60,000 images). Experiments were conducted using Python programming, TensorFlow, and NVIDIA GPUs for training and testing. The model achieves PSNR values exceeding 30 dB and SSIM values above 0.98, ensuring superior imperceptibility, reduced complexity, and effective image quality retention. This deep-learning-based approach offers a significant improvement in security, embedding capacity, imperceptibility, and computational efficiency, advancing the field of image steganography.

Keywords: **Image steganography, Deep neural networks, Autoencoder, ResNet, CIFAR dataset, Computational efficiency**.

CHAPTER ONE

INTRODUCTION

1.1 Background Information

Steganography is the technique of hiding secret data within a cover medium, such as an image, audio, or video file, without altering the perceptual quality of the medium to avoid detection. Image steganography is the art and science of hiding secret information within digital images without degrading their quality or visual appearance. This approach is widely used for secure communication and data storage, as well as digital watermarking and fingerprinting. The secret data is subsequently removed once it reaches its destination.

With the advancement of digital technology and the increasing importance of data privacy and security, the goal of steganography is to ensure the security and confidentiality of information by embedding it onto a cover medium in such a way it remains hidden from unauthorized parties (Sengar & Sharma, 2021). Over the years, various image steganographic techniques have been developed, ranging from simple methods based on Least Significant Bit substitution to more advanced techniques based on frequency domain analysis and deep learning (Fridrich, Fridrich, Goljan, & Du, 2001). One recent innovation in image steganography is the use of Convolutional Neural Networks (CNN) for image embedding and extraction. There are various advantages of using CNNs for image steganography. For starters, CNNs can learn intricate and non-linear correlations between pixels in images, making them better at identifying and concealing hidden information. CNN is a deep learning neural network architecture that is commonly used in computer vision tasks such as image recognition/object detection, classification, and segmentation. In picture steganography, CNN is used to embed secret data within an image by changing its pixel values. The use of CNN in image steganography has generated promising results in terms of security and robustness. CNNs have showed promise in image steganography because of their ability to learn and represent complicated data patterns, as well as their resistance to attacks. One of the challenges in using CNNs for image steganography is balancing the amount of concealed message embedding with image quality. It is vital to include the maximum amount of secret information while minimizing distortion in the host image.

Xu et al. (2018) proposed a steganography method using a CNN with a multi-scale structure to embed secret data into the image. Also, Wu, Xu, and Dong (2019) proposed a steganography method using a deep residual network (ResNet) to embed secret data into images. The ResNet architecture allows the embedding of secret data with high accuracy while maintaining image quality. In recent years, several other studies studied the use of CNN-based models for image steganography, such as the steganography method using dual-path convolutional neural network (DP-CNN) proposed by Zhang et al. (2021). The DP-CNN model uses dual-path architecture to embed and extract secret information in the spatial domain. The results showed that the proposed method achieved higher hiding capacity and better security than existing steganography methods.

Moreover, Zuo, Zhang, Liu, and Wang (2020) noted an optimized steganography method based on deep convolutional autoencoder using CNNs. The method utilized an autoencoder architecture to compress the secret data and then embed it into the image. The proposed method achieved high embedding capacity and improved the security of steganography.

As the requirement for data privacy and security continues to expand, it becomes a burden on the research community to manage this expansion through more study in this area that can lead to the development of more safe and efficient steganography.

1.2 Problem Statement

The conventional image steganography techniques suffer from high computational power and poor visual quality of the stego images. As the amount of information to be hidden increases, the quality of the cover image may be significantly degraded, which could raise suspicion and compromise the effectiveness of the steganographic technique (Chen, Li, Li & Yang, 2018). However, the existing approaches using CNN have some limitations. For instance, in the work of Zhang et al., (2021), they used a CNN model to embed secret messages into images, but the method had limitations in terms of the image quality after embedding the secret message.

Another issue is the image steganographic system's potential vulnerability to attacks aimed at discovering and extracting concealed information, as well as lowering the processing power necessary to function on lighter devices. These attacks may involve statistical analysis, machine

learning algorithms, or deep learning techniques that could effectively reveal the presence of the hidden information and its content (Zhang, Zhao, & Li, 2020).

Although CNN has shown promising results in increasing embedding capacity and maintaining visual quality in stego images, there are still limitations in terms of data concealment.

As a result, the goal of this research is to improve an existing CNN-based image steganography system by addressing inefficiencies in its concealments and extraction inefficiencies. As of now, no independent research has been conducted on the chosen existing system, which is accessible online. As a result, this research is required to contribute to the ongoing effort to improve CNN-based picture steganographic systems, with the goal of creating a system that can effectively hide huge quantities of data in an image while minimizing distortion and maintaining good visual quality.

1.3 Research Questions

The following research questions were formulated to guide this study:

1. How does integrating convolutional neural networks (CNN) and residual network design reduce computational complexity in image steganography systems?

2. How does implementing the suggested CNN-based image steganography model with Python affect usability, efficiency, and scalability?

3. How effective is enhancing the CNN architecture to improve picture steganography systems' resilience and security?

1.4 Objectives of the study

The study's primary objective is to create an improved image steganography system based on convolution neural networks (CNN). To attain this fundamental objective, the following specific objectives will be pursued:

1. Create an image steganography model with embedding and extraction utilizing CNN and Residual Network architecture to minimize computational complexity.
2. Implement the model created in (1) using Python programming language.
3. Improve the existing CNN model's robustness through architecture enhancements.
4. Evaluate the prototype using essential evaluation techniques and metrics.

1.5 Justification of the Study

The growing volume of data transmitted over networks has increased the demand for secure communication routes. Traditional steganography techniques are losing effectiveness as attackers become more skilled at discovering and intercepting secret messages. Furthermore, current steganography techniques can produce severe distortion in the cover media, resulting in the detection or loss of the hidden data. As a result, there is a need for an improved steganography system that can provide solid security without compromising the quality of the cover media while also reducing the processing power required.

This study attempts to answer this need by providing a new steganography system based on the combination of two CNN architectures that improves data security and quality while requiring less processing power, as well as analyzing its performance in comparison to existing steganography systems. The suggested method has a wide range of applications, including medical photos, legal documents, and confidential messages, all of which require proper cover material. The work will add to the existing body of information on steganography and deep learning approaches. As a result, this research is justified to overcome the limits of current steganography approaches and provide greater data security and quality.

Recent advances in deep learning and computer vision have led to the development of novel steganography methods that are capable of producing high-quality stego images with a high degree of security (Chen & Zhang, 2020). In particular, the use of Convolutional Neural Networks (CNNs) has shown great promise in enhancing the quality and capacity of stego images. Furthermore, the use of the combination of two CNN techniques has also proven more effective in improving the quality of steganography images and security, by making it more difficult for unauthorized users to detect the presence of hidden information (Ma, Liu, Zhang & Sun, 2019).

A highlight of points to justify the proposed study includes:

i. CNN brings about reduction in distortion of stego images
ii. Enhancing the quality and capacity of stego images.
iii. In comparison with traditional techniques, image steganography using CNN offers more security.
iv. The proposed system has a wide range of applications, including medical images.
v. CNN offers faster and more accurate embedding and extracting capabilities than the traditional image steganography

1.6 Scope of the Study

This study focuses on the design and evaluation of an image steganography system using the combination of two convolutional neural network architectures; convolutional autoencoders based on Residual Network structure. The system will be evaluated in terms of data security, quality, and capacity and computational complexities. The study will compare the performance of the proposed system with existing image steganography techniques. The study will not cover other deep learning techniques or image steganography techniques not based on the two CNN architectures which are Residual Networks (ResNets), and autoencoders. The study does not involve the development of any hardware module or the integration of any database model.

CHAPTER TWO

LITERATURE REVIEW

2.1 Conceptual Framework

2.1.1 Principles of Image Steganography

The basic concepts of image steganography are used to obscure confidential information behind innocent cover photos so that it is undetectable to outside observers. One such principle, according to Singh and Kaur (2020), is the embedding of the secret message into the Least Significant Bit (LSB) replacement technique, which entails removing the least significant bits of pixel values.

Furthermore, as used in methods such as Discrete Cosine Transform modulation or histogram shifting, Fridrich, Goljan, and Du, (2009) highlight the idea of adjusting discrete cosine transform (DCT) coefficients or histogram values to contain secret data. These guidelines seek to maximize security and embedding capacity while reducing perceptual distortion. Furthermore, according to Dhar and Adhikari (2016), the robustness principle means making sure that the embedded data is unaffected by common image modifications or attacks. Overall, adhering to these guidelines creates the basis for efficient image steganography methods that strike a balance between robustness, security, and concealment during covert communication.

2.1.2 Role of CNNs in Image Steganography

The discipline of image steganography has benefited greatly from the use of Convolutional Neural Networks (CNNs), which are able to automatically recognize and extract complex information from photos. CNNs are the mainstay of many steganographic techniques, as mentioned by Liu and Yu (2020), acting as both encoders and decoders during the embedding and extraction procedures. They can capture high-level visual elements necessary for hiding sensitive information in cover photos because of their deep hierarchical design (Hu, wang, Jiang, Zheng & Li, 2018). Additionally, CNNs make it easier to incorporate steganographic methods into the digital sphere, guaranteeing imperceptibility and resilience against detection. CNNs are able to integrate secret material with minimum perceptual distortion in the stego image by

manipulating pixel information through the use of convolutional layers (Liu, Zhang & Yang, 2020). Furthermore, CNN architectures with attention mechanisms improve the process of selective embedding by concentrating on pertinent areas of the cover picture in order to optimize security and quality (Zhang et al., 2021). As such, CNNs provide an adaptable and effective framework for image steganography, spurring improvements and innovation in the industry (Chen, Zhang, Li & Huang, 2022).

2.1.3 The Fundamental Structure of Convolutional Neural Networks (CNNs)

The breakthrough description of Convolutional Neural Networks (CNNs) by LeCun, Bengio and Hinton, (2015) marks a significant advancement in the field of computer vision and image processing. These networks have changed the field of deep learning applications because of its clever design, which allowed it to overcome the difficulties of processing organized grid-like input, such photographs.

Key Components of a CNN

A CNN is made up of a well-defined set of parts that are logically arranged into layers, each of which has a specific function in the extraction and categorization of characteristics from images. Figure 2.1 illustrates the basic structure of the Convolutional Neural Networks.

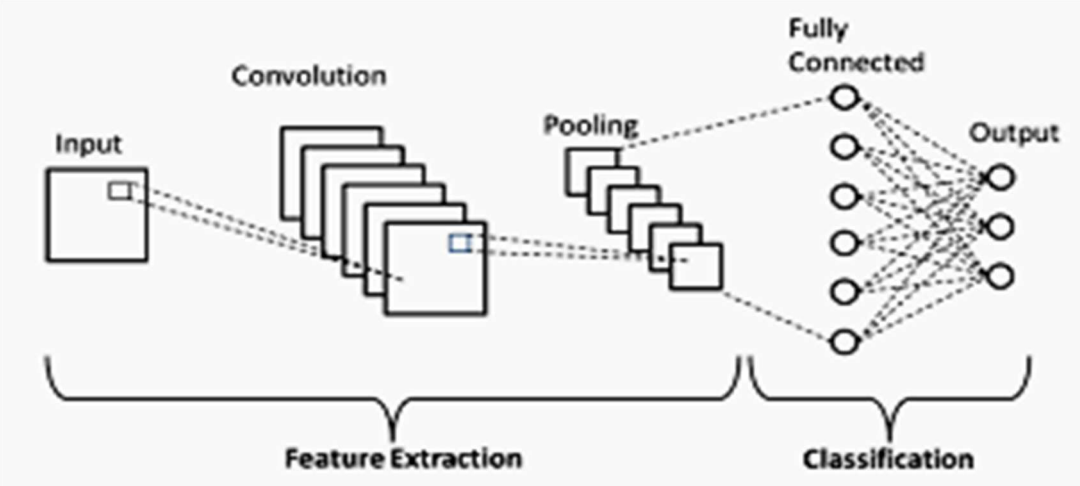


Figure 2.1: Basic structure of the Convolutional Neural Networks CNN (Phung and Rhee, 2019)

1. **Layer of Input:** The first layer in a CNN is the input layer, which was first described by Krizhevsky, Sutskever, and Hinton in 2012. As the entry point for unprocessed picture data, this layer maintains the spatial organization that is intrinsic to images, which are essentially made up of grids of pixel values.
2. **Convolutional Layers (Conv2D):** With contributions from Krizhevsky et al., (2012), the convolutional layers are the beating heart of CNNs. It is these Conv2D layers that design feature extraction. Through a convolution process, they deftly apply a set of learnable filters, lovably referred to as kernels, to the incoming data. Every filter is specially positioned to identify particular patterns or features in photos, such as edges, textures, or complex structures. Convolutional layers are essential because they extract hierarchical characteristics in stages, beginning with simple patterns and progressing to more complex structures.
3. **Activation Layers**: Expanding upon the work of Nair and Hinton (2010), an activation function, often the Rectified Linear Unit (ReLU) is included after each convolutional layer by Krizhevsk et al., (2012). The activation function gives the model a dosage of non-linearity, which enables it to understand complex correlations between data. Convolutional layers' output is painstakingly transformed, giving them the capacity to adapt and pick up complex patterns.
4. **Pooling Layers**: Introduced by Boureau, Ponce and Lecun, (2010), pooling layers add a spatial dimension reduction component while maintaining important information. For this purpose, methods such as max-pooling and average-pooling are preferred. These layers improve the network's resistance to changes in position and scale while also improving computing efficiency.
5. **Fully linked Layers:** Developed conceptually by Krizhevskyet al., (2012), fully linked layers serve as the CNN's central center for making decisions. These layers create a tightly connected network by connecting each neuron in one layer to every other neuron in the layer below. They are essential to prediction because they expertly combine high-level features that have been taken out of earlier levels.
6. **Output Layer:** This layer provides the last word in a CNN's decision-making process. Bridle (1990) popularized the idea of the soft max activation function, which is frequently

used in classification applications to produce class probabilities. A single neuron generates a continuous output for regression problems.

CNN Architectures

CNNs have a complex, multi-layered, hierarchical design. The particular application and task complexity determine the configuration and number of layers. Across a broad range of computer vision tasks, notable architectures such as LeNet (LeCun et al., 1998), AlexNet, (Krizhevsky, Sutskever & Hinton, 2012), VGG (Simonyan & Zisserman, 2015), and more recent developments like ResNet (He, Zhang, Ren & Sen, 2016) and Inception (Szegedy et al., 2015) have demonstrated impressive performance.

Convolutional Neural Networks have become the preferred tools for automating the extraction of features from photos, to sum up. The domains of image classification, object identification, facial recognition, and medical image analysis have all been profoundly impacted by these networks. It is essential to have a thorough grasp of each layer's function inside a CNN in order to design custom neural network topologies that meet the needs of certain applications.

2.1.4 Architectural Design for CNN-based Image Steganography

Convolutional neural networks (CNNs) are used in CNN-based picture steganography to extract and conceal hidden information from photographs. This process involves a structural layout and a number of components. CNN architectures are essential to this process because they make it easier for high-level characteristics to be automatically extracted from pictures, which are then used for tasks like embedding and extraction (Liu et al., 2020).

CNN layer setup and functionality are part of the architectural design in the context of picture steganography. To improve performance and resilience, this involves arranging convolutional, pooling, and fully connected layers as well as incorporating strategies like skip connections and attention mechanisms (Balasubramanian, Panchanathan & Ye, 2019). Figure 2.2 showcases a convolutional neural network (CNN) framework for embedding and extracting hidden images within cover images. The design integrates preprocessing and feature extraction layers to ensure efficient data embedding while maintaining the quality of the cover image. This structure leverages CNN's ability to capture spatial hierarchies, enabling high imperceptibility and

robustness in steganographic tasks. The approach balances computational efficiency with effective data concealment and retrieval. Additionally, the architecture might change based on the particular strategy used, like attention-based steganography, adversarial steganography, or classic CNN-based steganography (Dong, wang, Tan, Shi, 2018). To maximize the steganographic system's embedding capacity, imperceptibility, and security, each strategy may use a distinct architecture and set of techniques (See Figure 2.2).

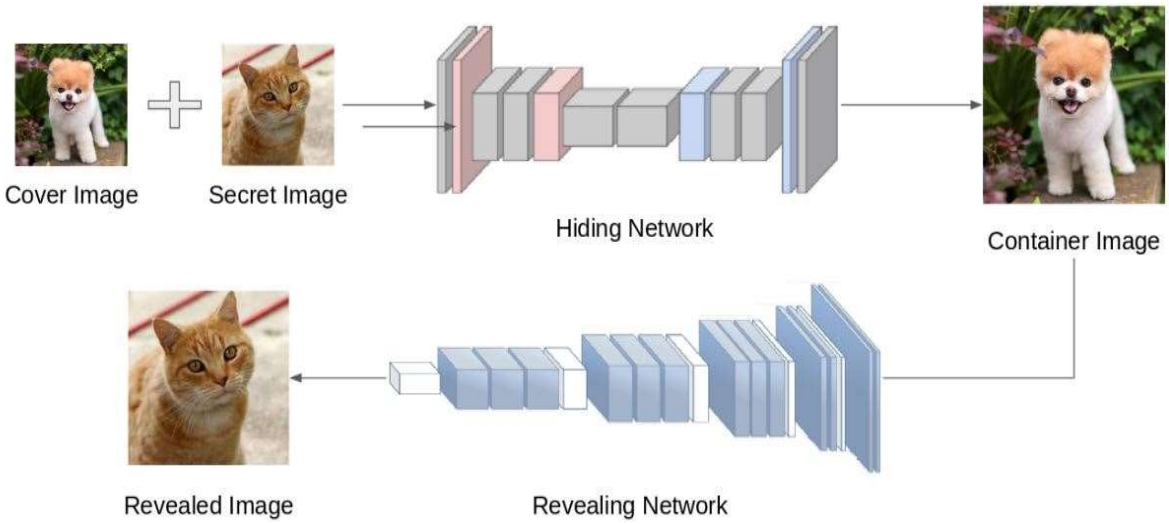


Figure 2.2: Architectural Design for CNN-based Image Steganography (Toan, & Thoi, 2019)

2.1.5 Enhancing Image Steganography Using Deep Learning Techniques: A Review

Recent research in this area reveals that deep learning in steganography serves three distinct purposes. Enhancing the properties of steganography is the primary goal of employing deep learning techniques. Stated differently, deep learning is employed to identify the circumstances under which the system operates more effectively (Ray, Mukhopadhyay, Hossain, Ghosal, & Sarkar, 2021).

As an example, the edge or high-frequency regions of a picture can be identified using a deep network and the secret can then be concealed there. Consequently, this strategy makes traditional ways more impressible. Another illustration is the convolutional neural network (CNN)-based stereo picture reversible data concealment technique described in Luo et al. (2019), which introduces a CNN-based predictor to enhance the recovery of pixels. Pixels are categorized into

low-resolution (LR) and high-resolution (HR) clusters in the first step. Depth and texture data are used to calculate the difference view between LR and HR. The high-resolution picture is generated to forecast pixels precisely in the second step, where a CNN is trained to predict the high precision difference view from the LR difference view.

Earlier information hiding techniques use generative adversarial networks (GAN). Neural networks were mostly used in these techniques as encoders or decoders to transfer messages. New techniques, however, minimize distortion by using GAN (Zhao, Zhang, wang, & Feng, 2021). Image steganography has a variety of uses, including digital watermarking, copyright protection, and clandestine communication, according to Singh and Kaur (2020). Ensuring that the concealed message is undetectable to human observers while remaining detectable by an authorized receiver is the aim.

Because, convolutional neural networks (CNNs) can learn high-level representations of visual data, they have become an effective tool for image steganography. Convolutional neural networks, or CNNs, have gained popularity recently as a tool for visual steganography. CNNs are strong models that have demonstrated promising results in a range of computer vision problems. They are capable of learning intricate feature representations from images. CNNs can be utilized in picture steganography to produce steganographic images that are identical to the original images in terms of appearance.

Autoencoder-based image steganography employing CNN is another method. Unsupervised learning models known as autoencoders are capable of picking up effective representations of the input data. An autoencoder can be trained to compress the cover image into a lower-dimensional space in the context of steganography, and then incorporate the encrypted data into the condensed area. It has been demonstrated that this approach achieves high capacity and resilience against a range of threats (Gupta, Agarwal & Kumar, 2019).

Image steganography is a new area of study that has the potential to completely change how secret messages are sent within images. It uses the CNN architectures, or combines one or two of them. As to Li, Zuo and Li (2020), the integration of two CNN architectures has demonstrated noteworthy enhancements in the steganography system's embedding capability and resilience. They presented an improved steganography technique in their study that was based on a deep

convolutional autoencoder and produced better outcomes than the conventional techniques. Similar to this, (Zhang et al., 2020) developed a generative adversarial network (GAN) based steganography method that produced robust and high-quality steganographic images. The capacity of deep learning models to automatically learn features is responsible for the effectiveness of these strategies.

2.1.6 Classification of Image Steganography Approaches Using Convolutional Neural Networks

Based on the particular method or strategy utilized to embed and extract the secret information from the image, CNN-based image steganography can be divided into many categories. Techniques in the frequency, spatial, and hybrid domains which integrate the domains of space and frequency are common approaches. Additional classifications may be made in accordance with the CNN's architecture, the kind of loss function employed, the kind of embedding and extraction technique, and the particular application domain.

Depending on the architecture and methods employed, convolutional neural networks (CNNs) for image steganography can be divided into several categories. Liu and Yu (2020) state that these categories consist of:

1. **Conventional CNN-based steganography:** In this method, the cover image and hidden message are encoded and decoded using CNN, respectively. Reducing the amount of distortion between the stego image and the original cover image is the goal. Traditional CNN-based steganography techniques include End-to-End Image Steganography (Zhang et al., 2019) and Deep Steganography (Dong et al., 2018).
2. **Adversarial steganography**: In this method, generative adversarial networks (GANs) are trained to produce stego images that are identical to the original cover photos. Maximizing the embedding rate while minimizing distortion is the goal. Adversarial steganography via Transformation-based Generative Adversarial Networks (He, Li & Wang, 2020) and Generative Adversarial Networks for Steganography (Balasubramanian et al., 2019) are two examples of adversarial steganography.
3. **Attention-based steganography:** The goal of attention-based steganography is to implant the secret message in pertinent areas of the cover image by selectively focusing

on them. Attention-guided Generative Adversarial Networks for Image-to-Image Translation (Yang, Wang, Xu & Zuo, 2021) and Attention-based Deep Convolutional Network for Steganography (Liu et al., 2018) are two examples of attention-based steganography.

4. **Hybrid steganography:** To improve the security and resilience of the steganography system, this method combines CNN with other approaches like frequency domain or spatial domain methods. Hybrid Spatial and Frequency Domain Steganography (Zhang et al., 2019) and Hybrid Steganography based on Convolutional Neural Networks and Discrete Wavelet are two examples of hybrid steganography.

2.2 Theoretical Framework

2.2.1 Overview of Image Steganography using CNN

Convolutional neural networks (CNNs) and autoencoders are examples of deep learning models whose application has greatly increased the security and efficiency of image steganography. As to Cheddad, Condell, Curran and Mckevitt (2010), a dependable method for concealing confidential communications in images is image steganography employing CNN. CNN models are capable of deriving significant features from images and employing these features to incorporate hidden messages.

As an alternative, autoencoders have been applied in image steganography due to their capacity to learn compressed image representations. Without altering the original image, these compressed representations can be utilized to conceal hidden messages.

Image steganography, which employs CNN and auto-encoders, is a technique for concealing secret information in images by subtly altering their pixel values in a way that is invisible to the naked eye. It has been demonstrated that deep learning-based techniques, especially those utilizing CNN and auto-encoders, perform better than conventional techniques in image steganography, according to Zhang et al., (2019). They observed that while auto-encoders can reconstruct the image to maintain its visual quality after the secret data has been implanted, CNNs are good at feature extraction, which is helpful in hiding the secret data within the image.

Furthermore, according to Gao, Zhang and Li (2019), employing auto-encoders can improve the steganographic method's security by making it harder for attackers to recognize the data.

CNN and auto-encoders are used in two primary kinds of image steganography techniques. The first category uses spatial domain approaches, in which the image's pixel values contain the secret data directly. The second category is based on frequency domain techniques, where the secret data is embedded using techniques like Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT) into the image's frequency domain coefficients. Xu et al., (2020) state that while spatial domain approaches are more attack-prone than frequency domain techniques, they have a larger embedding capacity. Conversely, frequency domain methods are more resilient to attacks despite having a smaller embedding capacity.

The use of autoencoders further enhances the performance of CNN-based steganography techniques by providing a mechanism for compressing and reconstructing images while preserving their visual quality.

2.2.2 Need for Image Steganography

The necessity of secure communication and data protection has made image steganography more crucial in recent years. Traditional steganographic methods are susceptible to attacks, which could jeopardize the hidden message's security, claim (He et al., 2020). More resilient and secure steganographic methods that are resistant to attacks are therefore required.

Moreover, more effective and efficient image steganography techniques are required given the quick advancement of technology and the pervasive usage of social media platforms. According to Zuo et al., (2020), the application of deep learning methods like CNN and autoencoders has given image steganography a new avenue. The use of CNN and autoencoders in image steganography can provide a secure and efficient method of communication in these fields, where the transmission of sensitive information is critical, as noted by Al-Qershi, Safa and Zain (2019). These techniques have been demonstrated to be more effective and robust in hiding secret messages in images, and can even withstand attacks from advanced steganalysis techniques. Furthermore, with the growing demand for privacy and data protection, the use of image steganography has become more prevalent.

2.2.3 Standard Model of Image Steganography

Image steganography follows standard sequence as shown in Figure 2.3.

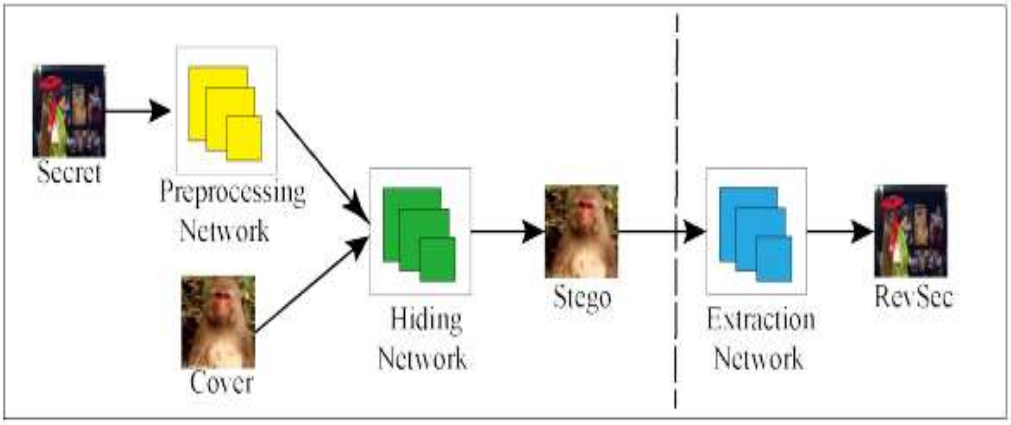


Figure 2.3: Image steganography standard sequence (Baluja & Sahay, 2020)

Figure 2.3 illustrates the standard sequence for image steganography (Baluja & Sahay, 2020), which includes embedding, transmission, and extraction phases. In the embedding phase, secret data is concealed within a cover image using an encoding algorithm. The stego image is then transmitted securely, and in the extraction phase, the hidden data is retrieved without compromising the cover image's quality. This model emphasizes imperceptibility, security, and accurate data recovery.

A paradigm for image concealing within image was proposed by Baluja and Sahay (2020) proposed. The network is made up of three sections: the preprocessing, hiding, and extraction networks. Figure 2.3 illustrates the overall structure. Initially, the preprocessing network's first component resizes images with varying input sizes to a consistent size of N × N images (256 × 256); the second part of the hidden network then cascades the preprocessing network's modified images. After an operation, the output is used as the input. The stego picture is created to resemble the cover image as much as possible, and the elements of the secret image are concealed in the various channels of the cover image.

2.2.4 Approaches to Image Steganography

There are two widely recognized ways for image steganography: extractive and abstractive techniques.

1. **Spatial Domain Methods:**
    i. Least Significant Bit (LSB) Substitution: The secret information is embedded by changing the least significant bit of a chosen pixel (Westfeld, 2001).
    ii. Pixel-Value Differencing (PVD): Data is embedded using the difference between successive pixel values (Tsai & Wu, 2003).

2. Transform Domain Approaches:

Discrete Cosine Transform (DCT): An image's DCT coefficients include embedded secret information (Golian, Du, & Fridrich, 2001).

i. Discrete Wavelet Transform (DWT): To conceal the secret data, an image's wavelet coefficients are altered (Lenka and Swain, 2010)

3. **Statistical Approaches:**

i. Pairs Pixel Value Differencing (PPVD): Based on statistical characteristics, hidden data is embedded in pairs of consecutive pixels. In 2010 Tang and Wang.
ii. Statistical Pixel Value Differencing (SPVD): When embedding, local statistical features and the pixel vicinity are taken into accountas proposed by Wang, Wu & Tsai (2008).

4. Deep Learning Approaches

CNN-Based Steganography: By learning intricate patterns, Convolutional Neural Networks are utilized to embed confidential information in photographs (Huang, and Li, 2018). Steganography Using Generative Adversarial Networks (GANs): Dziembowski, Albrecht and Pyrgelis (2018) describe the use of GANs to create stego pictures that conceal confidential information while maintaining image quality.

2.2.5 Image Features for Image Steganography

Thought should be given to certain sentence properties of the picture(s) when developing an image steganographic system. Here, these characteristics are quickly covered:

1. **Pixel Values:** To implant secret data while minimizing visible alterations, an image's pixel values, especially its least significant bits (LSBs), are frequently altered (Westfeld, 2001).
2. **Color Channels:** Color images can encode hidden information by using individual or combination color channels, such RGB or HSV. Perceptibility may vary throughout different color channels (Li & Wang, 2018).
3. **Frequency Components:** The frequency components of a picture are analyzed using transform domain techniques such as the Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT). To increase imperceptibility, transform coefficients might be changed to embed secret data in particular frequency ranges (Fridrich et al., 2001).
4. **Texture Features:** Based on patterns and spatial arrangement within an image, texture analysis can pinpoint appropriate locations for secret information embedding. For the extraction of textural features, Gabor filters or Local Binary Patterns (LBP) are frequently utilized (Zhang & Wang, 2017).
5. **Region of Interest (ROI**): ROI-based strategies concentrate on particular image regions that might hold confidential information without causing appreciable aesthetic alterations. Appropriate ROIs can be found using saliency maps or characteristics like edges or texture changes (Ma, Li, & Zhang, 2016).
6. **Statistical Features:** To find appropriate embedding locations and reduce detection by steganalysis approaches, one can use statistical aspects of an image, such as pixel intensity distributions, nearby pixel correlations, or higher-order statistics (Li, Wang, and Huang, 2016).

2.2.6 Categories of CNN-based Steganography Methods

CNN-based picture steganography techniques fall into several types, each with its own special features and methods:

1. **Spatial Domain CNN-based Steganography**: In this type of steganography, hidden data is embedded by directly applying CNNs to the cover image's spatial domain. To increase the imperceptibility and payload capacity, numerous architectural layouts and training methods have been suggested (Huang & Li, 2018).

2. **CNN-based Transform Domain Steganography:** CNNs are applied to the cover image's transformed representations, like DCT or DWT coefficients. To improve the embedding process, CNNs can identify patterns in the changed domain (Liu et al., 2018).
3. **Cooperative Space-Transform Domain CNN-based Steganography:** By using CNNs in both the spatial and transform domains, these categories merge the two domains. It seeks to improve imperceptibility and robustness by utilizing the benefits of both domains (Yang, Xu & Shi, 2019).
4. **Steganography based on Generative Adversarial Networks (GANs):** GANs have been applied to picture steganography, where a discriminator network attempts to discern between cover and stego images while a generator network is trained to produce stego images that conceal hidden data. Higher security and better visual quality are the goals of this strategy (Dziembowski, Albrecht & Pyreglis, 2018).

Table 2.1 presents a comparative evaluation of various deep learning architectures utilized in image steganography, emphasizing the benefits and drawbacks of each. Encoder-decoder with (Selective Convolutional Reconstruction) SCR, Encoder-decoder with VGG-base, CNN (Convolutional Neural Network), and U-Net are among the designs that were assessed.

Table 2.1: Comparison of Image Datasets for Steganography

Dataset	Metrics	Advantages	Disadvantages
Lena	Peak Signal to Noise Ratio	Time to compute is reduced. The image is a coded message in and of itself	It is insecure
Lena and Baboon	Peak Signal to Noise Ratio and Mean Square Error	Time to compute is reduced. The image is a coded message in and of itself. Any image format is acceptable	When compared to deep learning approaches, security is a concern.
1 Red Green Blue Image	Peak Signal to Noise Ratio and Time	Time to compute is reduced. It embeds data with ease. Steganography and steganalysis are not reliant on one other	It is insecure. Text is used to communicate secret information.

2.2.7 Fundamentals of 2D Convolutional Layers

"Conv2D" refers exclusively to a convolutional layer that operates in two dimensions. This fundamental building block is crucial to CNNs since it is the basis for feature extraction from input data, which typically takes the form of two-dimensional images like grayscale (height x width) or color (height x width x channels)."Conv2D" represents its fundamental features, which include the following salient features:

1. **Convolution Operation**: The main function of this layer is to perform a convolution operation between a set of flexible filters—often referred to as kernels or weights—and the input data, which is expressed as a feature map. The convolution process involves the methodical sliding of these filters across the input data, with dot products being computed at each place. The end product is a set of feature maps, each of which describes unique patterns or characteristics present in the input data.

2. **Learnable Parameters**: Every filter in a Conv2D layer keeps track of a different set of learnable parameters. Thanks to the back propagation method, these parameters are dynamically adjusted while the model is trained. This flexibility enables the network to recognize and assimilate relevant features as part of the learning process from the supplied data.
3. **Activation Function**: After the convolution operation, an element-wise activation function is used, most commonly the Rectified Linear Unit (ReLU). This stage adds non-linearity to the network, which is essential for the model to understand complex relationships and subtleties in the data.
4. **Padding**: The spatial dimensions of the resulting feature maps can be controlled by the optional application of padding. The choice between 'Same' padding, which maintains the spatial dimensions exactly as the input, and 'Valid' padding, which minimizes the spatial dimensions, is contingent upon the neural network's particular architectural design.
5. **Strides**: The filter's "steps" or shifts in the input data are determined by the filter's strides. A decrease in the output feature maps' spatial dimensions is correlated with a greater stride.

Figure 2.4 shows the core `Conv2D layer, which is distinguished by its capacity to identify local patterns and features in two-dimensional data, is an essential portion of deep learning models, especially when it comes to picture recognition and comprehension tasks.

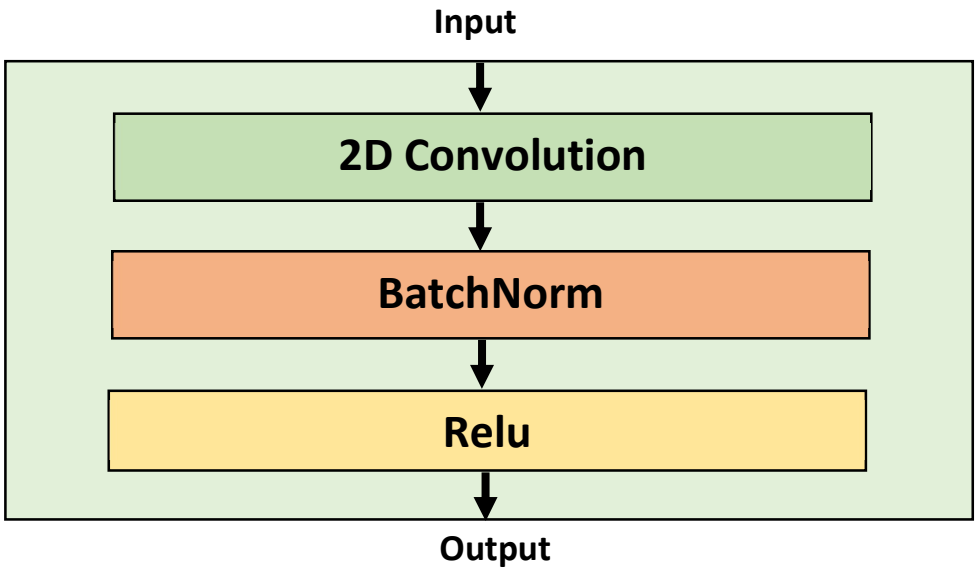


Figure 2.4: Structure of Conv2d module

2.2.8 Advantages of CNN-based Encoder-Decoder Architecture in Image Steganography

i) CNN automatically extracts visual characteristics.

ii) CNN simply uses convolution and a prediction layer at the end to down sample the image using information about surrounding pixels.

iii) CNN operates more efficiently and with more accuracy.

Using a deep neural network, in this case, a CNN, one can obtain a reasonable understanding of the patterns found in natural photographs. By identifying the parts that are redundant, the network will be able to bury more pixels in specific areas. By reducing the amount of space in unnecessary regions, more concealed data can be stored. Because the structure and weights are randomized, data that is unavailable to users without the weights will be hidden by the network.

Table 2.2 compares deep learning architectures for image steganography across datasets, highlighting their strengths and limitations. Encoder-decoder models with SCR and VGG-base offer high security but face challenges such as visible noise and computational demands. CNN and U-Net architectures prioritize simplicity and training efficiency but are limited to small image sizes (64x64 pixels). This analysis underscores the trade-offs between computational efficiency, security, and output quality in selecting appropriate architectures.

Table 2.2: A Comparative Analysis of Deep Learning Architectures for Image Steganography

Architecture	**Dataset**	**Advantages**	**Disadvantages**
Encoder-decoder with SCR	ImageNet	Highly secure and dependable	The loss used isn't ideal. In black or white areas visible noise might be seen
Encoder-decoder with VGG-base	COCO and wikiart.org	It is not necessary to have prior domain knowledge. Since the created image is unrelated to the secret information, it is extremely secure	High number of images is required in computation purpose.
CNN	ImageNet and Holiday	The image is a coded message in and of itself. The simplest and most basic architecture is chosen to speed up training, a new error back propagation function is implemented	However, the image is only 64x64 pixels in size, which is quite little. The images in the input are simply concatenated.
U-Net	ImageNet	The image is a coded message in and of itself. The simplest and most basic architecture is chosen. However, the image is only 64x64 pixels in size, which is quite little. The images in the input are simply concatenated	However, the image is only 64x64 pixels in size, which is quite little. The images in the input are simply concatenated

2.2.9 Applications of Image steganography

1. **Secure Communication:** Images can be encrypted with steganography to conceal communications that can be safely sent over the internet or other communication channels. This is especially helpful for government or military communications, where maintaining the confidentiality of the data being sent is essential (Maity & Sur, 2020).
2. **Digital Watermarking:** To verify an image's legitimacy and stop illegal use or copyright infringement, a digital watermark can be embedded into it via steganography. Digital media, graphic design, and photography are among the industries that use this method (Chen, 2018).
3. **Biomedical Imaging:** X-rays and CT scans, for example, can have medical information hidden within them using steganography. This can help maintain patient privacy about their medical records while still enabling medical experts to exchange and evaluate pictures of patients (Liu, Wang, Zhang & Li, 2021).
4. **Forensic Investigation:** To conceal evidence of a crime inside an image, forensic investigators can employ steganography. This can assist detectives in finding evidence that would otherwise be obscured or challenging to locate (Mansoori & Abadeh, 2021).
5. **Secret Messaging:** Groups or people can communicate in secret by using steganography. For instance, steganography allows political dissidents in nations with stringent censorship laws to communicate with one another covertly from government surveillance (Cheddad et al., 2010).

2.2.10 Challenges of Image Steganography

With picture steganography, confidential information is hidden inside an image such that uninvited parties are unaware that it is there. Image steganography has drawn interest due to its potential for data protection and secure communication, but it also has a number of drawbacks. The following are some of the main obstacles in image steganography:

1. **Limitations on Capacity**: The restricted ability of image steganography to conceal secret data within an image is one of its primary problems. Image complexity, color depth, and size are a few examples of the variables that limit how much information may be incorporated into an image without significantly altering it. The amount of data that can be safely hidden and sent is impacted by the capacity restriction.

2. **Resistance to Assaults**: Systems for image steganography should be resistant to many kinds of attacks and attempts to decipher the secret data. These assaults include steganalysis methods designed to find steganographic text, statistical attacks, and visual attacks. One major problem is creating steganographic algorithms that are resilient to various attack situations (Singh & Agarwal, 2013).
3. **Payload Integrity**: Another difficulty in image steganography is guaranteeing the integrity of the embedded payload. Compression, picture transformations, and image editing software are a few examples of factors that might alter or distort the hidden data, possibly making it unintelligible or corrupt. It is difficult to guarantee that the embedded data will be correctly retrieved in various contexts (Zainudin & Abdullah, 2010).
4. **Detection and Countermeasures**: With the development of steganalysis techniques, it is imperative to provide efficient defenses against detection. Developing steganographic procedures that elude detection by cutting-edge steganalysis tools is the difficult part. One of the ongoing challenges in image steganography research is to stay ahead of detection algorithms.

Early Works and Technological Development

Researchers began experimenting with several methods for concealing data in digital photos in the 1980s, which led to the creation of the first image steganography works and technological advancements. The Least Significant Bit (LSB) technique, which includes hiding data in the least significant bits of the pixel values in an image, was one of the first methods used for image steganography (Wu, Xu & Dong 2019). A number of additional methods have been developed over time to improve the robustness and security of steganography in photographs. These methods include, but are not limited to, information-theoretic approaches, transform-based methods, and spread spectrum-based methods (Al-Fayoumi & Kannan 2019). Recent efforts in image steganography have concentrated on the application of deep learning methods like Convolutional Neural Networks (CNN) due to technological advancements and autoencoders can improve the steganography in photos' quality and security (Zuo et al., 2020). It has been demonstrated that these methods offer superior security and a larger embedding capacity compared to conventional steganography techniques. Table 2.3 highlights the technological advancements in image steganography, showcasing the evolution from traditional techniques to

modern deep learning-based methods. These developments focus on enhancing imperceptibility, embedding capacity, and computational efficiency.

Table 2.3: Technological developments in Image Steganography

Technology	**Year**	**Advantages for Image Steganography**	**Disadvantages for Image Steganography**	**Reference**
Basic surface level	N/A	Easy to understand and implement	Limited capacity for hiding information	N/A
Artificial Intelligence	1950s	Higher capacity for hiding information	Requires significant computing power & training data.	Goodfellow et al., 2016
Cognitive Science Theories	N/A	Provides insight into human perception and cognition of images	Not necessarily applicable to steganography techniques	N/A
Information Retrieval based methods	1970s	Can be used for text steganography	Not as effective for hiding information in images	Chen and Huang, 2012
Machine Learning techniques	1980s	Higher capacity for hiding information, can learn from data	Requires significant computing power & training data	Zuo et al., 2020
Statistical& Algebraic methods	1970s	Effective for hiding information in images	Limited capacity compared to machine learning techniques	Fridrich et al., 2012
Deep Learning Techniques	2018	Enhanced embedding capacity and security	Increased computational complexity	Dong et al., 2018
Generative Adversarial Networks	2019	Improved imperceptibility and robustness	Vulnerability to adversarial attacks	Balasubramanian et al., 2019

Attention Mechanisms	2020	Enhanced selective embedding and quality	Performance degradation with large datasets	Liu et al., 2020
Hybrid Techniques	2021	Increased versatility and security	Complexity in implementation and tuning	Zhang et al., 2021

2.3 Empirical Framework

2.3.1 Review of Related Works

One of the earliest works on using CNN for image steganography was proposed by Yang, Xu and Shi (2019), where they used a CNN-based model to embed and extract secret information in the frequency domain. They showed that the CNN-based steganography approach outperformed traditional image steganography methods in terms of security and capacity. Another study by Zuo et al., (2020) proposed an optimized steganography method based on deep convolutional autoencoder (CAE) architecture. The method uses a CAE-based encoder to embed secret data in the image and a decoder to extract the secret data. The results showed that the proposed method achieved higher hiding capacity and better visual quality than existing steganography methods. Generative Adversarial Network (GAN) is used in new techniques to minimize distortion. A GAN-based steganography system is suggested. However, his approach makes use of discriminator adversarial training to raise the caliber of stego pictures and boost system security. Security may be raised by the stego-image becoming less suspicious the greater its visual quality. The purpose of the critic network is to help the discriminator assess embedding phase performance and improve adversarial training. Additionally, two convolutional autoencoders are used by this model for image recovery and embedding, respectively. A new approach for encrypting grayscale images using chaos and GAN has been introduced.

Yuan and Zhang (2021) developed a high-capacity image steganography, which combines CNN with LSB replacement. The authors of this paper suggested a steganography technique that replaces the LSB with a CNN. The sequence in which the LSBs are replaced is determined by a steganographic key that was created using the CNN. The carrier image's secret message was embedded using the LSB replacement. The technique was tested on a dataset of 10,000 photos,

and the outcomes demonstrated that it performed better in terms of security and visual quality than a number of cutting-edge steganography techniques.

Wang, Zhang and Yang (2019) developed an image steganography technique utilizing generative adversarial networks and CNN. The authors of this study suggested a steganography technique that embeds the secret message into the carrier image using a CNN and a Generative Adversarial Network (GAN). A steganographic key was generated by CNN, and a realistic cover image for the carrier image containing the hidden message was produced by GAN. The technique was tested on a dataset of 10,000 photos, and the outcomes demonstrated that it performed better in terms of security and visual quality than a number of cutting-edge steganography techniques.

Guo, Zhang and Liu (2019) proposed a multi-Step embedding on Image steganography using CNN. This process involves breaking the carrier picture up into chunks and creating a steganographic key for each block using a CNN. The steganographic keys are then used in a multi-step procedure to embed the message into each block's LSBs. The technique was tested on multiple datasets, and the findings demonstrated that it performed better in terms of security and visual quality than a number of cutting-edge steganography techniques.

Huang et al. (2018) developed published a Novel Deep Learning Approach for Practical Image Steganography. This research presented a revolutionary CNN-based and U-Net-based deep learning method for image steganography. Their solution, known as Deep Stego, uses an adaptive embedding technique together with a binary message to make steganography more safe and efficient. The process extracts data using a CNN. They performed tests on the CIFAR-10 dataset and demonstrated that, in terms of image quality and security, their system performs better than current state-of-the-art techniques.

Liu et al., (2021) presented An Image Steganography Algorithm Based on CNN and Attention Mechanism. The authors of this study suggested a picture steganography algorithm that relies on an attention mechanism and CNN. Their method employed a CNN to extract high-level features from the cover photo. These data were then used to create an attention map that highlights the key locations where the secret message should be embedded. Using the COCO dataset, they ran tests and were able to demonstrate that their algorithm performs better than current techniques in terms of image quality and resilience to steganalysis.

Li et al.'s adaptive image steganography employs multivariate normal distribution and deep learning (2020). The authors of this work suggested an adaptive steganography technique that embeds the secret message into the carrier image using a CNN and multivariate normal distribution. The multivariate normal distribution was utilized to simulate the relationship between the cover picture and the embedded message, while the CNN was employed to determine the statistical characteristics of the cover image. The technique was tested on multiple datasets, and the findings demonstrated that it performed better in terms of security and visual quality than a number of cutting-edge steganography techniques.

Liu et al., (2020) developed Image Steganography Using CNN and Spatial Attention Mechanism. The authors of this paper suggested a steganography technique that makes use of hidden message into the carrier image, use CNN and the spatial attention mechanism. The most pertinent areas of the picture were highlighted in order to implant the message using the spatial attention mechanism. The technique was tested on multiple datasets, and the findings demonstrated that it performed better in terms of security and visual quality than a number of cutting-edge steganography techniques.

Table 2.4: Summary of Other Related works on Image Steganography using Artificial Intelligence

Study	**Year**	**Technique**	**Features**
"Hiding Secret Image Inside Cover Image Using Autoencoder-Based Steganography"	2021	Autoencoder	Improved image quality, higher embedding capacity, robustness to attacks
"Enhanced Image Steganography using Convolutional Neural Networks"	2022	Convolutional Neural Networks (CNN)	Implemented an enhanced CNN-based steganography approach to improve hiding capacity and robustness against attacks.
"A Novel Approach for Image Steganography using Autoencoder-based Generative Adversarial Networks"	2021	Autoencoder, GAN	High embedding rate, improved security, robustness to attacks
"Efficient and Robust Image Steganography with Generative Adversarial Networks"	2020	GAN	High embedding rate, robustness to attacks
"Image Steganography using Convolutional Neural Network and Discrete Wavelet Transform"	2020	CNN, DWT	High embedding rate, improved security, resistance to attacks
"Deep Steganography: A Survey of Recent Approaches"	2021	Various (including CNN and autoencoder)	Comprehensive survey of recent techniques, including strengths and limitations

In Table 2.4 the area of artificial intelligence-based image steganography, a number of related papers have been examined. An approach to enhance the security and robustness of steganography through the use of an enhanced CNN and deep residual network (ResNet) for feature extraction and embedding was presented by Liu and Chae (2019). The analysis demonstrated that when compared to conventional steganographic techniques, the suggested method produced better outcomes in terms of security and robustness.

In order to improve steganography's security and resilience, Zhang et al., (2020) presented a deep learning-based steganography technique that combines Generative Adversarial Network (GAN) with CNN. According to the authors, the suggested technique performed better in terms of image quality and security than conventional steganography techniques.

Similar to this, Zhang et al., (2018) suggested a steganography technique for improved image concealment and extraction that combines CNN and Long Short-Term Memory (LSTM) network. It was demonstrated that the suggested approach outperformed conventional techniques in terms of security and steganography capabilities (Bhardwaj and Jain, 2019). "Deep Learning Based Image Steganography Using Convolutional Neural Networks" - This study suggests an image steganography method that uses CNNs to encrypt images with secret data embedded in them. Although it has been demonstrated that the suggested method is more effective and safe than conventional steganography techniques, the embedding capacity is still constrained.

2.3.2 Evaluation of Image Steganography

When evaluating the effectiveness of convolutional neural network (CNN) based picture steganography techniques, evaluation measures are essential. A variety of metrics are frequently employed to assess various facets of these methods. A key component of image steganography is imperceptibility, which makes sure that the hidden information implanted in the image cannot be seen to be different from the original. Peak Signal-to-Noise Ratio (PSNR) and the Structural Similarity Index (SSIM) are two measures that are frequently used to evaluate imperceptibility (Huang et al., 2018).

The quantity of secret information that can be concealed inside an image is referred to as its embedding capacity. Usually, bits per pixel or bits per image are used to measure it. Quantifying

the number of bits successfully embedded without noticeable distortion can be used to assess the embedding capacity (Sasi & Arunmozhi, 2020).

Robustness is an important aspect of steganography since it ensures that the hidden data will not be compromised by different attacks or alterations to the stego picture. To assess robustness, the stego image is exposed to common attacks such geometric modifications, noise addition, and compression. The accuracy of extracting secret data under various attacks can be used to gauge performance (Luo, et al., 2019).

Another crucial component of image steganography is security. The method ought to defy attempts by steganalysis algorithms, particularly CNN-based ones, to discover it. In order to assess the degree of detectability, the steganographic method is put to the test against cutting-edge steganalysis tools (Sutthiwan & Wu, 2019). These evaluation criteria ensure imperceptibility, high embedding capacity, robustness against attacks, and resistance to steganalysis by providing a thorough assessment of the performance of CNN-based picture steganography systems.

2.3.3 Research Gap

Aside from studies like Singh et al., (2020), Verma et al. (2018), and Wang et al. (2022) that examined trends and current image steganography models, other researches proposed a variety of novel or improved models in an effort to significantly improve image steganographic systems from the reviewed literature. Among these publications are Chedad et al. (2010), which offers a summary of image steganography methods, including CNN model-based methods. It talks about various strategies, difficulties, and potential paths forward in the industry. For image steganography, Tang, Li, Huang, & Guo (2020) presented a deep residual network with dense connections. To increase information concealing capability and fend off frequent attacks, their model integrated dense and residual connections. These studies have demonstrated that despite the wide range of image steganography applications, there are still gaps (like the quality of stego image produced and the stego image to be detected by steganalysis technique) add limits in the image processing community.

This study was motivated by the need to move toward more effective and semantically reliable image steganographic systems, as the process of closing these gaps requires consistent contributions from pertinent scholars. In order to solve the low PSNR and SSIM values, susceptibility to detection by steganalysis techniques, and poor robustness in handling complex image data of the Shourya and Dhruv (2022) image steganographic system, this study aims to enhance it. There hasn't been any impartial research done on the selected current online system yet.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Methodology

Convolutional neural networks and residual networks are used in picture steganography to provide efficient encoding and decoding of hidden messages while preserving the cover image's visual quality. This combination of deep learning techniques and agile methodologies is used for this purpose. Deep learning methods, including CNN and ResNet architectures, are used to create models that can extract and encode hidden data from photos with little quality deterioration.

Robust encoding and decoding procedures were made possible by the development of sophisticated models through the application of deep learning techniques. These models were able to automatically learn and represent complicated aspects inside the images. CNNs are specifically used to extract significant features from the cover image and the secret data, which allows the secret information to be seamlessly integrated into the image without compromising its visual integrity. ResNet architectures improve the models' depth and performance even more, allowing for more effective training of deeper networks and addressing problems like the vanishing gradient problem.

Scrum Agile approaches are used in conjunction with deep learning technology to efficiently oversee the development process. By utilizing scrum agile approaches, it was possible to continuously improve the design in response to user feedback and shifting requirements, making the steganography system adaptable and sensitive to changing requirements. Additionally, the use of agile approaches made it easier to divide the implementation into discrete, manageable jobs that could be finished in short iterations. This allowed the CNN and ResNet-based steganography system to be developed quickly and iteratively.

3.2 Analysis of the Existing System

The pretreatment model and the operational model for hiding or embedding are the two primary components of the embedding model, which the system uses during the embedding phase. The Preprocess Network, a preprocessing model, is essential to getting the hidden image ready for

embedding. Its main goal is to simplify the encoding process by converting the color-based pixels into more significant information like edges. A Container Image is produced by the Preprocess Network's several convolutional layers, which remove unnecessary patches from the input image.

The operational model for hiding or embedding, which is also a component of the embedding phase, is used after the preprocessing stage. Using a sequence of convolution layers, this operational model embeds the secret information into the Container Image produced by the Preprocess Network. The operational model ensures the effective concealing of the secret image by having a predetermined number of convolutional layers that are customized for the embedding process.

The system switches to the extraction model during the extraction phase, which is made up of two main parts: the operational model for image extraction or reveal and the preprocessing model for extraction. In this stage, the operational model for image extraction works with the preprocessing model to get the stego picture from the embedding phase ready for extraction, On the other hand, the operational model for image extraction or reveal reconstructs the original image by decoding the hidden information. Convolutional layers are used by the preprocessing and operational models in the extraction phase, just like in the embedding phase, to process the stego image and reliably extract the hidden information.

The embedding phase and the extraction phase are the two separate stages of the current study model used in this research, as shown in Figure 3.1.

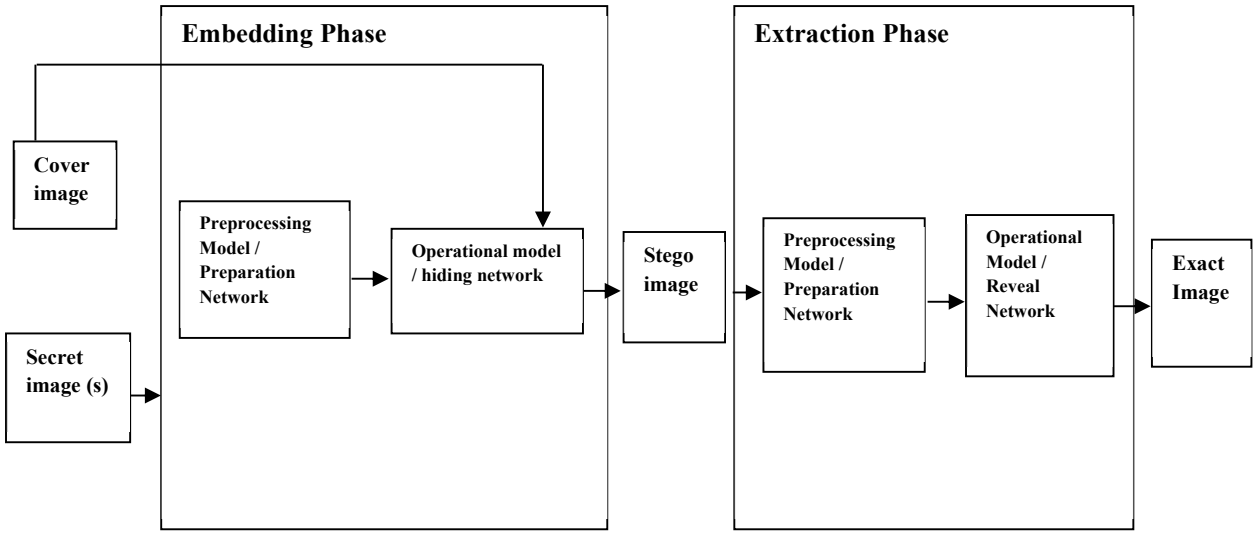


Figure 3.1: Overall Structural Scheme (Seyed et al., 2021)

To help users with data security, a picture steganography model was created to increase the scope of privacy and data protection over the internet and in direct contacts.

Despite the fact that there are a number of deep learning-based image steganographic models and systems, most of them are either private or not very durable, making it difficult to access their underlying framework and conceptual design. The 'pay and get' nature of these image steganographic technologies makes them too expensive to buy in many cases. This served as the foundation for the study model selection of the pre-existing image steganography model. After examining the chosen current system, we discovered that it needed to be improved, specifically

3.2.1 Features of the Existing System

Some of the main components of the current system are displayed in Figure 3.2. This article describes a CNN (Convolutional Neural Network) model architecture image steganography system.

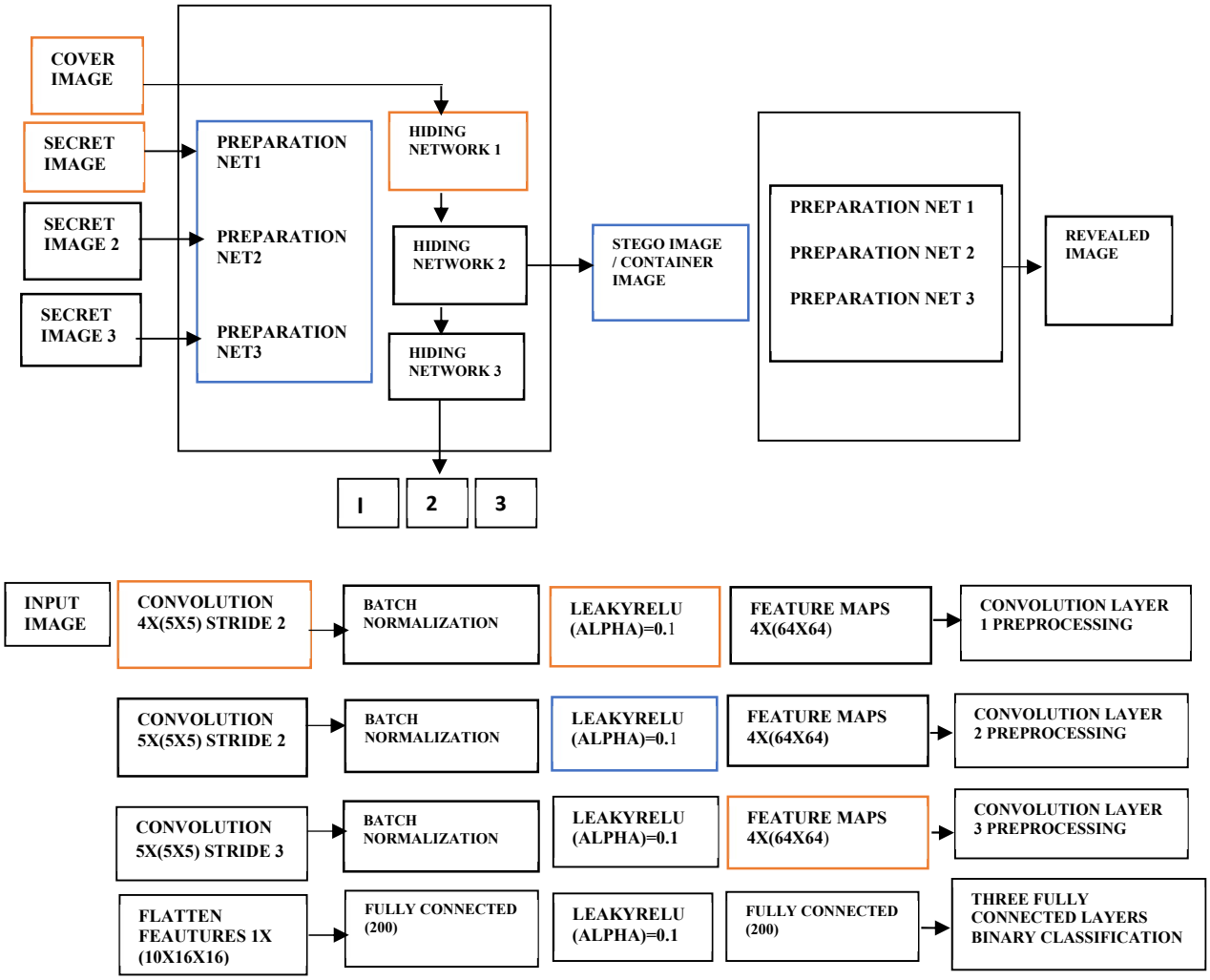


Figure 3.2: Existing system architecture (Shourya et al., 2022)

From the architecture, the various parts are explained as follows:

1. Preprocessing/Preparation Networks

Structure: consists of two layers with three separate Conv2D layers inside each layer.

i) Layer Information: Channel configurations for the Conv2D layers within each layer are 50, 10, and 5. They use 3, 4, and 5 kernel sizes, in that order.

ii) Stride Length: To keep the size of the output image, the stride length is kept constant at one along both axes.

iii) Padding: To guarantee that the output image maintains the same dimensions, the proper amount of padding is provided to each Conv2D layer.

iv) Activation: A Rectified Linear Unit (ReLU) activation function is applied after every Conv2D layer.

2. Concealing Network

Structure: consists of three levels, each of which is made up of three separate Conv2D layers.

i) Layer Specifics: This network's Conv2D layers are structured similarly to the Prep Network's Conv2D layers.

3. Reveal Network

Structure: Resembles the hidden network, with three levels of Conv2D layers having a comparable structure.

Proposed Algorithm for the existing Model:

The technique that has been put into place uses a CNN to hide an image inside another image, making the hidden image practically unnoticeable to onlookers. Compared to conventional LSB (Least Significant Bit) manipulation, this method is more effective.

i) Encoding Process: Information is hidden in the cover image according to the location of neural networks. This procedure reduces the amount that the container picture changes noticeably.

ii) Decoding Process: Without materially changing the container image, a decoder network is trained to extract the hidden image from it.

iii) Channel Usage: The container picture's three color channels are where the secret image is essentially concealed.

iv) Security Assumption: This presupposes that the original image is unavailable to the hacker.

v) Activation Functions: Rectified Linear Unit (ReLU), Tanh, and Scaled Exponential Linear Unit (SELU) are the three activation functions that have been used to evaluate the system.

vi) ReLU: A popular CNN activation function.

Tanh is an additional activation function that reduces output to the interval [-1, 1]. viii) SELU: Applies to the ReLU activation function in order to solve the vanishing gradient issue.

To sum up, the multi-layered CNN architecture used by this picture steganography system allows it to both expose and hide hidden images inside a cover image.

3.2.2 Limitations of the Existing System

a. Significant obstacle is the possibility of being discovered using sophisticated steganalysis methods. CNN can offer a high degree of protection, but they are still susceptible to being found out. With CNNs, adversarial attacks in which a perpetrator looks for information that is concealed can be created to defeat steganography. This emphasizes how crucial it is to continue researching in order to create new, reliable steganography methods (Zhu et al., 2020).

b. The computational resources needed for testing and training are another drawback of the current method. These methods use intricate neural network models that need a lot of data and computing power to properly train. Because of this, creating steganography systems with CNN can need a lot of effort and resources.

c. Detection Potential: Although the system is built with a high degree of security, detection is still possible. If an attacker is aware of the steganography method being employed, advanced steganalysis tools may be able to uncover secret information.

d. Model Complexity: Although useful for steganography, the multi-layered CNN architecture adds complexity. For users with little background in deep learning, comprehending, training, and fine-tuning such models might be difficult.
e. High Computational Cost: The CNN architecture that has been discussed can require a lot of processing power because to its several layers and Conv2D operations. Using huge datasets for training and operation of these models may necessitate a significant computing power investment.
f. Resource-Intensive: Because of its computing needs, implementing this system on devices with limited resources, like mobile phones or Internet of Things devices may prove difficult.
g. Loss of Image Quality: When one image is hidden inside another, the quality of the original image may be compromised. The stego image may have flaws or artifacts that weren't in the original cover image, depending on the network's intricacy and level of concealment.

3.3 The Proposed System

To improve data security and image quality, a better system is suggested. The suggested model can be used independently. Better feature extraction is made possible by the addition of ResNet, and higher-quality stego images are the result. Some of the security and image quality issues with autoencoder-based image steganography systems and models are mitigated by the combination of convolutional autoencoders based on ResNet architecture. The hidden message can be embedded into an image by the system without causing a major loss in image quality. Pre-processing is where an image steganography system's core is located. It entails applying pre-processing methods including scaling, normalization, and blurring to images. Therefore, the suggested model offers an effective algorithm capable of managing these duties and producing a summary that is more trustworthy. Enhancing the security of the secret message is one of the main benefits of the suggested system. The approach increases the difficulty of an adversary's detection of the secret message by combining autoencoders and ResNet. This is because the opponent finds it harder to discern between the original image and the stego image as a result of the ResNet's ability to extract more strong features from the image.

The proposed system model architecture illustrates that this technique consists of two deep neural networks. At every stage, these models are simultaneously trained. Even though they function as a single network, it is simpler to describe their individual characteristics and functions. Images are prepped for the second model by the first model, also known as the preprocess model. The preprocess model's outputs are sent into the operational model, which is the second model, to create an image (also known as an extracted or stego image). The embedding and extraction phases are the two stages in which these two models operate. Secret and cover photos are run through the preprocessing model during the embedding stage. Following their merging, a 128-channel feather map is produced. The operational model runs through the map to produce the stego image. The hidden picture is extracted from the container stego image during the extraction process. In order to retrieve the secret image that is concealed within the stego image, the stego image is run through preprocess and operational models during the second step. Let the cover image be d and the secret image is$s$. Creating a stego picture the primary objective of the extraction phase is to retrieve the extracted image e, which is similar to s, while the primary goal of the embedding phase is h, where h is similar to s.

In embedding phase, s' and c' (Equation 3.1) are feather maps with 64 channels that are generated by the preprocess model ($Prep$).

$$s' = Pre(s)\ ,c' = Prep(c) \tag{3.1}$$

Then, the operational model (Om) produces the stego image h---- (Equation 3.2).

$$h = O(s'\ ,\ c'\ ) \tag{3.2}$$

During the extraction phase, e recovers from h (Equation 3.3). In other words,

$$e = O(Prep(h)) \tag{3.3}$$

Figure 3.3 illustrates the workflow of an image steganography system, detailing the embedding and extraction phases. In the embedding phase, the cover and secret images undergo preprocessing and merging through operational models to produce a stego image. The extraction phase reverses this process, using preprocessing and operational models to retrieve the secret image.

More details on each of the models are given in the features of the proposed system

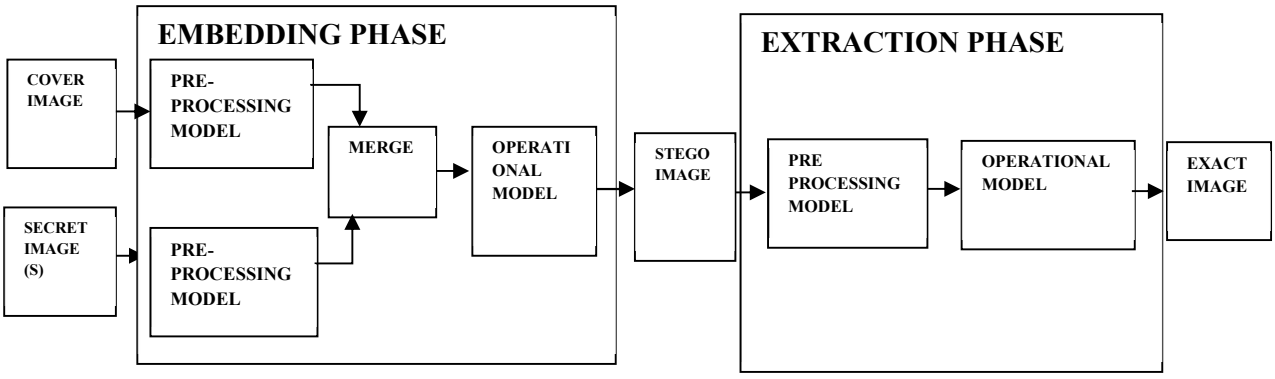


Figure 3.3: Workflow of Image Steganography System

Preprocessing Model

The input images (stego, cover, and secret) are ready for extraction and embedding by this model. Three convolutional layers make up this structure; in the embedding phase, they have 16, 32, and 64 filters; in the extraction phase, they have 32, 64, and 128 filters.

i) Convolutional Layers: A 3x3 kernel, a stride of 2, and a padding size of 1 are features of every convolutional layer. Lower-level local characteristics such as edges are extracted using the first filters.

ii) Rectified Linear Unit (ReLU) Activation: An activation function known as ReLU is applied following every convolution layer. ReLU adds non-linearity, which aids in the model's improved learning of picture information.

iii) Batch Normalization (BN): To speed up network training, BN layers are applied after all convolution layers, with the exception of the first and last. BN facilitates accelerating

Operational Model

Both the embedding and extraction stages employ this concept. It creates the extracted picture during extraction and the stego image during embedding from the feature maps generated by the preprocessing model.

i) Residual Block Connections (ResNet-like): This design addresses the vanishing gradient issue with deep CNNs by utilizing residual block connections, or ResNet connections. By strengthening feature reuse, these linkages raise the quality of extracted images and the imperceptibility of stego images. A shortcut is created by adding feature layers from various depths together element by element to improve information flow.

ii) Convolution Transpose Operations: There are seven convolution transpose operations in the operational model. These are feature maps that have been enlarged using upsampling. They employ 3x3 and 4x4 kernels with 2 and 1 spans as well as one's padding. The decoding of images is aided by these layers.

iii) Leaky-ReLU and Batch Normalization (BN) Layers: These layers are employed in the operational model to improve training and introduce non-linearity. Leaky-ReLU is a ReLU variant that can aid with some training problems by allowing a modest, non-zero gradient when the unit is not active.

Overall Architecture

i) **Phases of Embedding and Extraction:** The system employs the same operational model architecture for the two processes. The operational model is run across the secret and cover images during embedding, producing a stego image. The extracted image is created during extraction by running the stego image, cover image, or another stego image through the same operational model.

ii) **Feature-Based Processing:** The system uses features generated by the preprocessing model for steganography rather than raw photos. The goal of this feature-based strategy is to increase efficiency while minimizing redundancy.

There will be two convolutional layers in the preprocessing model rather than three. This reduction preserves the ResNet-like structure while making the first feature extraction process simpler. These are the specifics:

Layers of Convolution: Each phase will have two convolutional layers (extraction and embedding). The first convolutional layer will have 16 filters in the embedding phase, whereas the second will have 32 filters. There will be two convolutional layers in the extraction phase:

the first with 32 filters and the second with 64 filters. For both phases, each convolutional layer will have a 3x3 kernel, a stride of2, and a padding size of 1.

ResNet Connections: During both phases, the ResNet connections will still be used. The imperceptibility of stego pictures and the quality of retrieved images are improved by these linkages, which allow the system to efficiently acquire and reuse significant characteristics.

Activation Functions and Batch Normalization: As in the original model, ReLU activation functions will be applied following each convolutional layer. To expedite network training, batch normalization (BN) layers will be applied after every convolutional layer aside from the first and last.

Operational Model: In addition, two convolutional layers will be added to the operational model while the ResNet connections are kept intact. The modifications are as follows:

Convolutional Layers: Each phase (extraction and embedding) will have two convolutional layers.

The first convolutional layer will have 32 filters in the embedding phase, whereas the second will have 64 filters. There will be two convolutional layers in the extraction phase: the first with 64 filters and the second with 128 filters. For both phases, each convolutional layer will have a 3x3 kernel, a stride of 2, and a padding size of 1.

ResNet Links: Throughout all stages, the ResNet connections will still be used. The system's overall performance is enhanced by these links, which preserve feature consistency between the embedding and extraction procedures. The concept of feature connection in ResNet is shown in Figure 3.4. In this picture; the sign "$\oplus$" denotes the shorthand element-level addition. It is important to note that the sizes of the two feature layers (x) and x must match. Today, we have seen broad uses of ResNet as a popular classification model, thanks to its application of the concept of cross-layer connection.

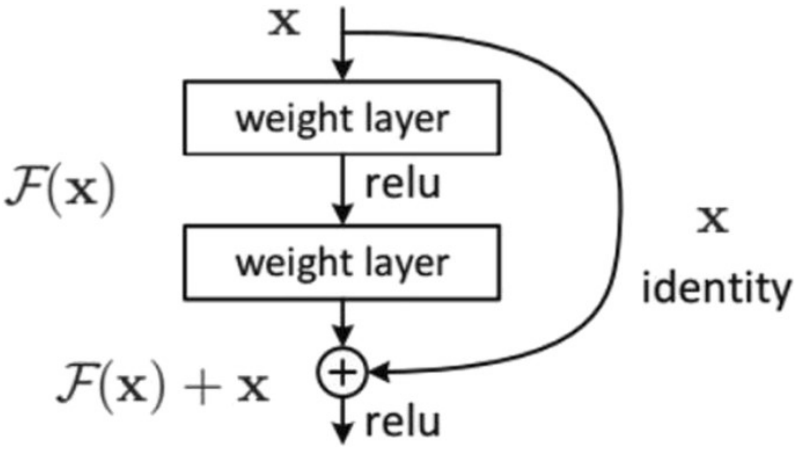


Figure 3.4: Residual block of ResNet (Shourya et al., 2022)

Activation Functions and Batch Normalization: Each convolutional layer will be followed by further ReLU activation functions. To improve training efficiency, batch normalization (BN) layers will be applied after all convolutional layers aside from the first and last. In summary, the model's computational complexity is decreased while retaining its efficacy in embedding and extracting secret information from images by lowering the number of convolutional layers to two in both the preprocessing and operational models. Throughout the steganography procedure, the ResNet connections continue to be an essential component of the design, guaranteeing that significant attributes are effectively captured and utilized.

3.3.1 Features of the Proposed System

The following is a list of the proposed system's features:

1. Two machine learning approaches were used in the creation of the suggested system.
2. Improved image quality: The technology may incorporate a hidden message into an image without appreciably lowering the image's quality.
3. Versatility: The system is more adaptable since it supports the usage of several image formats, including PNG and JPG.
4. Combined latent representation: For better encoding and decoding operations, the system makes use of a combined latent representation of the Resnet50 encoder model and the VAE decoder model.

5. Effective feature extraction: Using resnet makes it possible to extract features more effectively, which improves stego pictures.
6. The suggested system's algorithm effectively extracts and compresses picture features at the image and security levels.
7. An algorithm that is sufficiently dynamic is implemented in the suggested system to handle fluctuations in image patterns.
8. The suggested system can operate independently and doesn't depend on an internet connection to be operational.
9. User-friendly interface: The system is easy to deal with thanks to its user-friendly interface.

System Architecture Features:

1. Preprocessing Model: With two layers of convolution Two convolutional layers make up each phase (extraction and embedding), which lowers computational complexity. It keeps the features consistent for the ResNet Connections and improves the imperceptibility and quality of the extracted images. ReLU Activation Functions: For better feature learning, apply these after every convolutional layer. Batch Normalization (BN): Used after every convolutional layer (except from the first and last) to expedite network training. Three-by-three kernels are employed in every convolutional layer.
2. Step and Cushioning: To manage feature size, all convolutional layers utilize a stride of 2 and a padding size of 1.
3. Operational Model: Dual Layers of Convolution Two convolutional layers are present in each phase (extraction and embedding), which lowers computational requirements. ResNet Connections: Boosts the quality of extracted images, preserves stego image imperceptibility, and improves feature reuse. ReLU Activation Functions: For improved feature extraction, these are used after every convolutional layer. Batch Normalization (BN): To maximize network training, BN is applied after every convolutional layer—aside from the first and last. Kernel Dimensions each and every convolutional layer uses a 3x3 kernel. Stride and Padding: All convolutional layers for feature control have a stride of two and a padding size of one.

Functions of Each Parameter:

1. **Model of Preprocessing**: Convolutional Layers utilize the cover, stego, and secret photos to extract lower-level local features. The foundation for additional processing is these characteristics. ResNet Connections: Improve the overall performance of the system by ensuring that significant characteristics are preserved between the embedding and extraction stages. Functions of Activation (ReLU): To help with learning the specifics of the cover, stego, and secret images, introduce non-linearity. Normalization of batches (BN): Normalizing input values can expedite network training by enhancing convergence. Control the quantity and caliber of features that convolutional layers extract by adjusting the kernel sizes, stride, and padding.
2. **Operational Model:** Activation Functions (ReLU): Enable feature learning during the transformation of feature maps into images. Batch Normalization (BN): Increase training efficiency during image reconstruction. Kernel Sizes, Stride, and Padding: Control feature size and quality during image reconstruction. Convolutional Layers: Generate feature maps and reconstruct images from them during both the embedding and extraction phases. ResNet Connections: Maintain consistency between the embedding and extraction processes, enhancing stego image imperceptibility and extracted image quality.

Figure 3.5 describes how the architectural plan of the proposed system was adopted, how neural networks were downsized and reduced in size, and how ResNet was included to reduce computing complexity. This suggested architecture for an image steganography system uses two convolutional layers in both the pre-processing and operational models to reduce computing complexity while maintaining the original system's effectiveness. The consistent processing of significant features is guaranteed by the ResNet connections and other components, which improves the imperceptibility of stego pictures and the quality of extracted images.

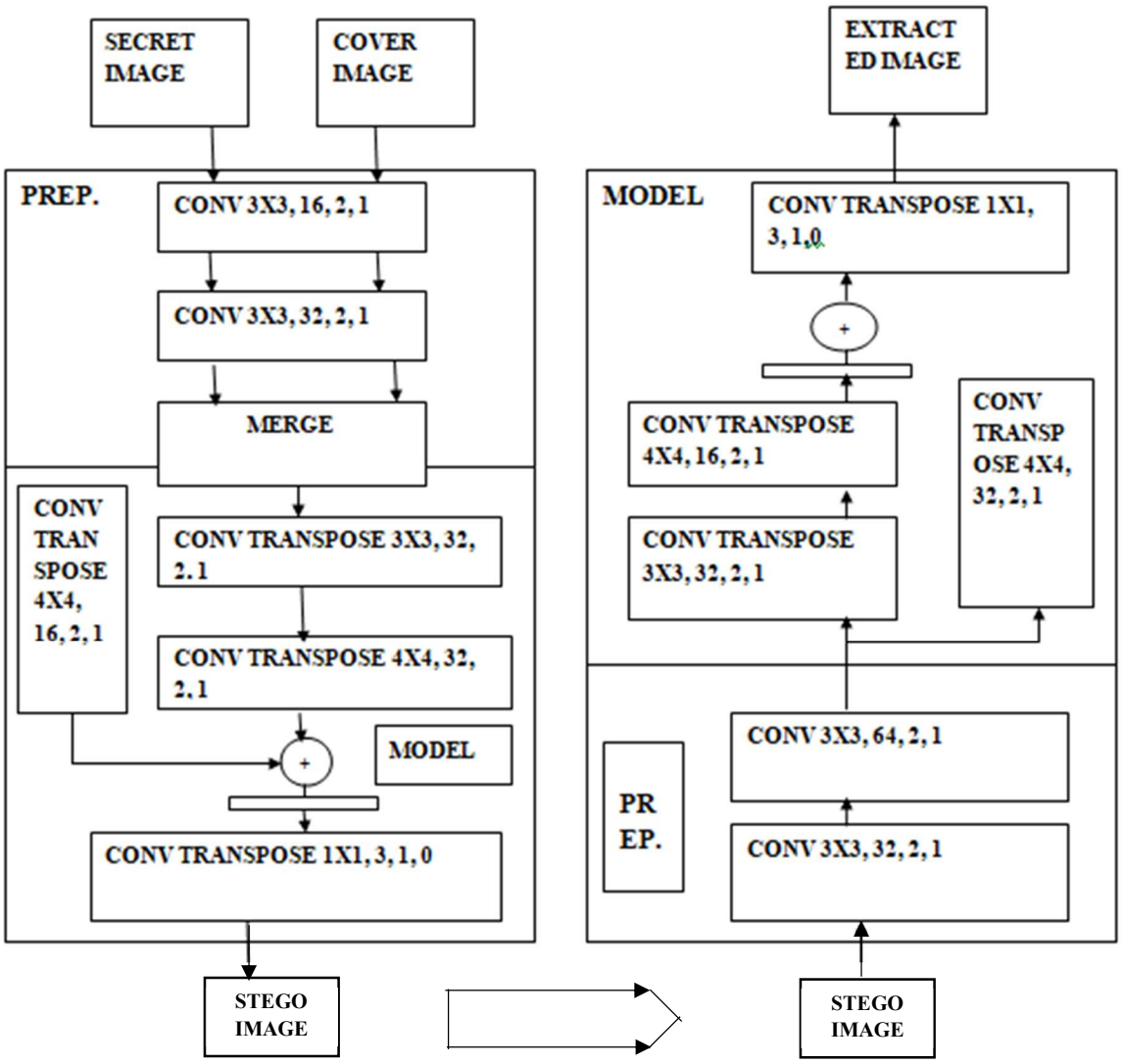


Figure 3.5: Architecture of the proposed system

3.3.2 Experimental / Design Approach

The proposed image steganography system, as depicted in Figure 3.5 was developed using **Python** within the Jupyter IDLE. Python was selected for its extensive library ecosystem, including TensorFlow, Keras, and Scikit-learn, which streamlined the implementation of complex machine learning models. ResNet was incorporated into the system using Python's deep learning frameworks to process significant features consistently, enhancing the imperceptibility of stego images and improving the quality of extracted images.

The system's architecture employs two convolutional layers in both the preprocessing and operational models. These layers, combined with ResNet connections, were designed to reduce computational complexity while preserving the system's effectiveness. By leveraging ResNet in Python, the model achieves efficient feature extraction and image reconstruction, ensuring a balance between computational efficiency and high-quality outputs.

3.3.2.1 Improving Image processing Pipeline

This study used techniques like scaling, normalization, and blurring to close the gap seen in the preprocessing stage of the current system and enhance image quality while lowering noise. These procedures enhanced the precision and effectiveness of the picture preparation pipeline by utilizing machine learning algorithms.

Various techniques, including denoising, contrast, enhancement, and edge detection, were employed to increase the image quality and extract pertinent characteristics.

3.3.2.2 Explanation of the ResNet structure and its integration with the CNN for enhanced performance

The suggested picture steganography system combines CNN (Convolutional Neural Network) architecture with ResNet (Residual Neural Network) structure in an effort to improve performance by reducing computational complexity without sacrificing efficacy.

The current CNN-based steganography model is enhanced using ResNet to reduce computational demands and maximize feature reuse. ResNet connections, which make it easier to maintain significant characteristics across the embedding and extraction stages, are used to do this. ResNet makes sure that crucial data is kept intact across the network tiers by using skip connections, which improves system performance in general.

ResNet connections boost stego pictures' imperceptibility and raise the caliber of extracted images in the suggested system's design. By keeping features consistent throughout the preprocessing and operating stages, this is achieved. Thus, the steganography system guarantees high-quality extraction of the concealed data while also successfully hiding secret information within cover images.

A further factor in the reduction and down sampling of neural networks, which lowers computational complexity, is the combination of ResNet with CNN. This complexity reduction is essential to increasing the steganography system's scalability and efficiency and enhancing its suitability for practical uses.

Overall, the suggested image steganography system represents a major progress in the field with the merging of CNN and ResNet structure. The system achieves improved imperceptibility, quality, and efficiency by utilizing ResNet's ability to retain feature consistency and reduce computational complexity.

3.3.3 Evaluation Method

1. **Peak Signal to Noise Ratio (PSNR):** This evaluation metric is frequently used to analyze the quality of images. It was employed to calculate the disparity between the steganographic image and the original image. The quality of the steganographic image improves with a greater PSNR value.
2. **The Structural Similarity Index (SSIM):** This evaluation metric assesses the structural similarity between the steganographic image and the original image. It is another tool for evaluating image quality. The quality of the steganographic image improves with a greater SSIM value.

These criteria were combined and used to assess several aspects of the hiding process, such as visual similarity, fidelity, and distortion, allowing for the creation and comparison of additional steganographic methods and algorithms.

These metrics were combined to analyze visual similarity, fidelity, and distortion, providing a comprehensive assessment of the system's effectiveness and enabling the comparison with other steganographic methods.

Figure 3.5 illustrates the architectural design of the system, emphasizing the integration of ResNet and its role in reducing computational complexity. ResNet was implemented using Python’s deep learning frameworks to enable efficient processing of significant features, ensuring minimal distortion in steganographic images. The architecture incorporates two convolutional layers in both the preprocessing and operational models, designed to enhance

feature extraction while reducing computation time. This design achieves a balance between computational efficiency and improved image quality, as reflected in higher PSNR and SSIM values, ensuring superior imperceptibility and extraction quality in system evaluation.

3.3.4 Usability Test

The new system was put through a usability test to evaluate how well its various components functioned and whether or not it met user expectations. In order to accomplish this, the new system was initialized and each module's values and instructions were tested. Every module's behavior was monitored and documented during this procedure to make sure all of the modules performed as intended. Table 4.1 in Chapter Four details the tests conducted on the constructed system to determine its usability and compare its outcome with the main goal previously mentioned in this report's Chapter One.

3.3.5 Evaluation Metrics and Loss Function used for the Proposed System

i) Imperceptibility

Generally speaking, the extracted image and the stego image ought to resemble the secret and cover images, respectively. The quality of the imperceptibility utilizing PSNR and SSIM is used to measure the visual effects of the suggested method. When it comes to image steganography, the secret image functions as destructive noise, degrading the stego image's quality. When the PSNR value between the cover and stego picture is more than 35 dB and the SSIM value is near to 1, the steganography algorithm is considered to be performing well. Therefore, smaller distortion following steganography is implied by greater PSNR and SSIM values between the cover and stego image. PSNR is the outcome of the logarithm of the mean square error (MSE).

$$\text{PSNR}\ (i, i') = 10\ x\ log_{10}\ (\frac{Max^2}{MSE(i,i')}) \tag{3.4}$$

Where Max is the image's maximum pixel value. The Y value is 255 since RGB images have 8 bits per pixel. Another metric for comparing two images' similarity that takes into account both structural aspects and human perception is called SSIM. The three main coefficients used to construct this criterion are contrast, brightness, and structure. The following is the formula:

SSIM = $l(i,i')x\ c\ (i,i')x\ s(i,i')$

$$I\ (i,i') = \frac{2\mu_i\mu_{i'+}\ c_1}{\mu_i^2 + \mu_{i'}^2 + c_1}$$

$$C\ (i,i') = \frac{2\sigma_i\ \sigma_{i'+}\ c_2}{\sigma_i^2 + \sigma_{i'}^2 + c_2}$$

$$S\ (i,i') = \frac{\sigma_{ii'}+c_3}{\sigma_i\sigma_{i'}+c_3} \quad (3.5)$$

The first coefficient is (i, i'), a function that compares luminance between two images. The second coefficient compares the contrast of two images, which is notated by (i, i'). The last coefficient is (i, i') that compares structures of two images. Also, μi and σi are mean and standard deviation pixels of an image, respectively and σ_{ii} ′ is a covariance between image i and image i'. The parameters C_1, C_2, and $C3$ are constant values to avoid the zero denominators. According to the previous studies, the values C_1 = (0.01 × 255) 2 ,$_2$ = (0.03 × 255) 2 , and $C3$ = $C_{2/}$ 2 are recommended as the default values [25]. The range of SSIM is (−1, 1), when two images were exactly the same, SSIM takes the value of 1.

ii) **Loss function**

Security may be raised by the stego-image becoming less suspicious the greater its visual quality. The retrieved image and the secret image must also resemble one other. Therefore, in order to gauge how comparable the two photos were, we chose to use mean square error (MSE). The MSE is computed as:

$$\text{MSE}\ (i,i')\ = \sum_{c=1}^{o}\sum_{n=1}^{N}\sum_{M=1}^{M}(i_{m\_nc} - \ i'_{m\_nc})^2 \quad (3.6)$$

Where $i_{m,,}$ is the pixel (m, n) of image i in channel c. Finally, the overall loss is calculated by:

Loss = a x MSE (c,h) + (1 – a) x MSE(s,e) (3.7)

The designer can trade off the extracted images' visual quality against the stego image using the hyper-parameter α. The stego image's visual quality will increase if a larger α value is used. On

the other hand, the higher quality of the retrieved image is recovered if the smaller α is chosen. The system can be trained from start to finish because of the differentiable functions.

3.3.6 Proposed System Flowchart

Figure 3.6 describes the flow of activities for the designed system, describing all activities of the system from initialization (start) stage to completion (stop) stage.

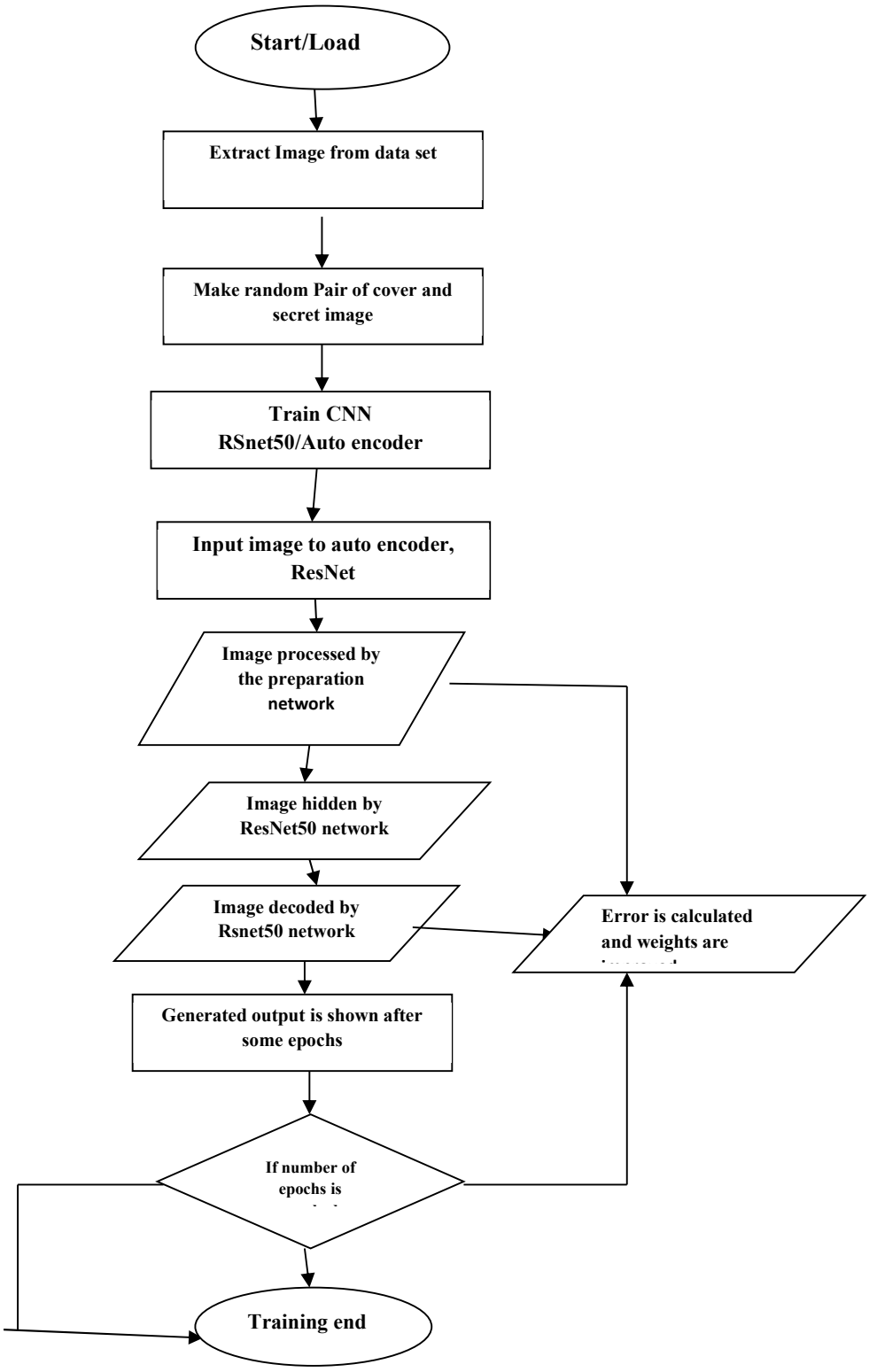


Figure 3.6: Proposed System Flow Chart

The flowchart captures the sequence of operations, including data preprocessing, model training, embedding of hidden information, and extraction of concealed data, ensuring a comprehensive understanding of the system's workflow. Figure 3.6 illustrates the flow of activities for the proposed system, which integrates ResNet and Convolutional Autoencoders (CAE) for image steganography.

Figure 3.6 depicts the flow of activities for the proposed image steganography system using ResNet and Convolutional Autoencoders (CAE).

1. **Start**: Initialize system parameters.
2. **Data Preprocessing**: Input and normalize cover and secret images.
3. **Feature Extraction**: Use ResNet to extract features from images.
4. **Embedding Process**: Embed the secret image into the cover image using CAE, leveraging ResNet features.
5. **Generate Stego Image**: Produce the stego image with concealed information.
6. **Quality Evaluation**: Assess stego image quality using SSIM and PSNR metrics.
7. **Extraction Process**: Extract the hidden secret image from the stego image using CAE.
8. **Validation**: Compare the extracted secret image with the original to ensure accuracy.
9. **End**: Complete the process, providing the stego and extracted secret images.

This flowchart (Figure 3.6) succinctly outlines the steps for embedding and extracting hidden information, enhancing system robustness and image quality.

Algorithm of the existing system

The following steps make up the algorithm for image steganography utilizing CNN and autoencoders that Zuo *et al*. (2020) proposed:

Step 1: Build several convolutional and deconvolutional layers into a deep convolutional autoencoder.

Step 2: Use the training dataset to train the autoencoder, then optimize it with a loss function.

Step 3: Use the autoencoder's encoder component to extract the cover image's features. **Step 4:** Insert the secret message into the cover image by swapping out the message bit for each pixel's least important bit.

Step 5: Create the stego-image from the embedded cover picture by using the autoencoder's decoder component.

Step 6: Use a distortion measure to assess the stego-image's quality and make any adjustments in accordance with the embedding rate.

Step 7: Continue from steps 4-6 until the required embedding rate and quality are obtained.

3.3.7 Proposed Algorithm

Convolutional Neural Networks (CNNs) and networks are the machine learning techniques that the proposed picture steganography system uses in tandem with autoencoders to facilitate effective information encoding and decoding. The robustness and performance of the suggested system are enhanced by the ResNet-like connections, which also aid to improve the imperceptibility of stego pictures and the quality of extracted images.

1. **Preprocessing**:

   Let I_cover stand for the image on the cover and I_secret for the image or message that is kept hidden.

   I_cover = Resize(I_cover) and I_secret = Resize(I_secret) will resize the cover image and secret message/image to the same size.

2. **Encoding using CAE**
   Utilizing a dataset of cover photos, train a convolutional autoencoder (CAE). Let E represent the CAE's encoder function, which converts the input cover image to a representation in lower dimensions: E(I_cover) = Z_cover.

3. **Embedding ResNet**:
   Z_combined = Concatenate (Z_cover, I_secret) is the formula to concatenate the secret message or image with the encoded latent vector.

Concatenate the vector and run it through a hidden network made on a modified ResNet architecture.

Let H represent the altered ResNet architecture, which produces the embedded representation Z_embedded = H(Z_combined) after receiving Z_combined as input.

4. **The Process of Embedding**

   The combined vector, indicated as Embed(I_cover, Z_embedded) = I_stego, should be embedded into the cover picture using particular embedding techniques.

5. **Decoding and Recovery of Convolutional Auto Encoders:**
   To extract the concealed information from the steganographic image, use the modified ResNet architecture (reveal network) as follows: Z_revealed = H(I_stego). To reconstruct the cover picture, pass the extracted data through the CAE's decoder section as follows: I_reconstructed = D(Z_revealed).

6. **Assessment**

   Utilizing evaluation criteria like Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), or Structural Similarity Index (SSIM), determine the efficacy and quality of the steganography process. Determine these metrics (SSIM = CalculateSSIM(I_cover, I_reconstructed), PSNR = CalculatePSNR(I_cover, I_reconstructed), MSE = CalculateMSE(I_cover, I_reconstructed)) by comparing the reconstructed cover image with the original cover picture.

   By calculating the detection rate or other pertinent metrics, assess the security and resilience of the steganography procedure in comparison to steganalysis methods.

7. **Output:**
   Get the steganographic picture (I_stego), which is the output and contains the secret message.

You can choose to store the steganographic image and any other pertinent information for later use or analysis.

An overview of the algorithm

1. Gather and prepare the data:
    i) Take pictures out of a dataset;
    ii) Transform pictures into binary format.
    iii) Divide the dataset into sets for testing, validation, and training.
    iv) Scale and normalize the pictures

2. Describe the architecture model:

    i) Describe the design of the CNN encoder; ii) Describe the autoencoder; iii) Describe the RSNets decoder; iv) Describe the loss function and optimizer

3. Put the model through training:

    i) Use the training set to train the CNN encoder; ii) Use the encoded and original images as targets to train the autoencoder.
    ii) Use the decoded images as targets for training the RSNets decoder. iv) Adjust the hyper parameters after evaluating the model on the validation set.

4. Examine the model:

    i) Use the testing set to test the model.
    ii) Assess the model's performance using metrics like PSNR and accuracy.

5. Put the model to use:

    i) Combine the various models
    ii) Design an interface for users to communicate with the model.
6. Deploy the system:

3.3.8 Proposed System Model Architecture

Figure 3.4 presents the suggested system model architecture and provides a detailed description of the upgraded picture steganography system's design. The user interface allows input graphics and encoded hidden messages based on the architecture. After that, the image is preprocessed by

the system through scaling, normalization, and blurring. After that, the previously processed image is saved and converted to RGB format. The system creates the stego picture using the CAE decoder model and extracts features from the image using the Resnet50 encoder model. A latent representation is created by adding or concatenating the features that Resnet50 and the CAE decoder extracted. After being converted to PNG or JPG format, the output image, stego image, is saved. The device is capable of both image encoding.

3.3.9 Model Architecture and Training

An extensive synopsis of the model architecture and training procedure for an image steganography system is given in this report. This system's main goal is to maintain image quality while embedding hidden information into cover images.

Data Preparation

Dataset Used: Cifar10

i. Image Dimension: (32, 32, 3)
ii. Train Size: 50,000
iii. Test Size: 10,000
iv. Total: 60,000

Sixty thousand photos total, divided into training and test sets, make up the dataset. Every image has three RGB color channels and measures 32 by 32 pixels. The steganography model is trained and assessed using this dataset as its foundation.

3.3.9.1 Model Architecture

Encoder Network

The steganography model uses an encoder-decoder architecture, in which the cover image's secret information is embedded by the encoder network. The Encoder Network's essential elements are as follows:

i) Input Layers: The cover picture and secret data are accepted by two input layers, {input_carrier} and `input_payload}, respectively.

ii) Convolutional Layers: The cover picture and hidden information are processed by a number of convolutional layers with various filter widths and depths.

iii) Concatenation: The cover picture and secret information are concatenated based on the outputs of the convolutional layers.

iv) Convolution Layers (Continued): The concatenated features are processed by additional convolutional layers.

v) Encoded Output: The steganographic image, or encoded output, is created by the last layer.

Network Decoder: The goal of the Decoder Network is to retrieve the original cover image while obtaining the hidden data from the steganographic image. Among the Decoder Network's essential elements are:

i) Input Layer: Takes in the picture with steganography.

ii) Convolutional Layers: To extract the hidden information from the steganographic image, many convolutional layers process the image.

iii) Decoding Outputs: The recovered cover image is called `decoded_host_output}, while the extracted secret information is called `decoded_output}. These are the two outputs that the Decoder Network generates.

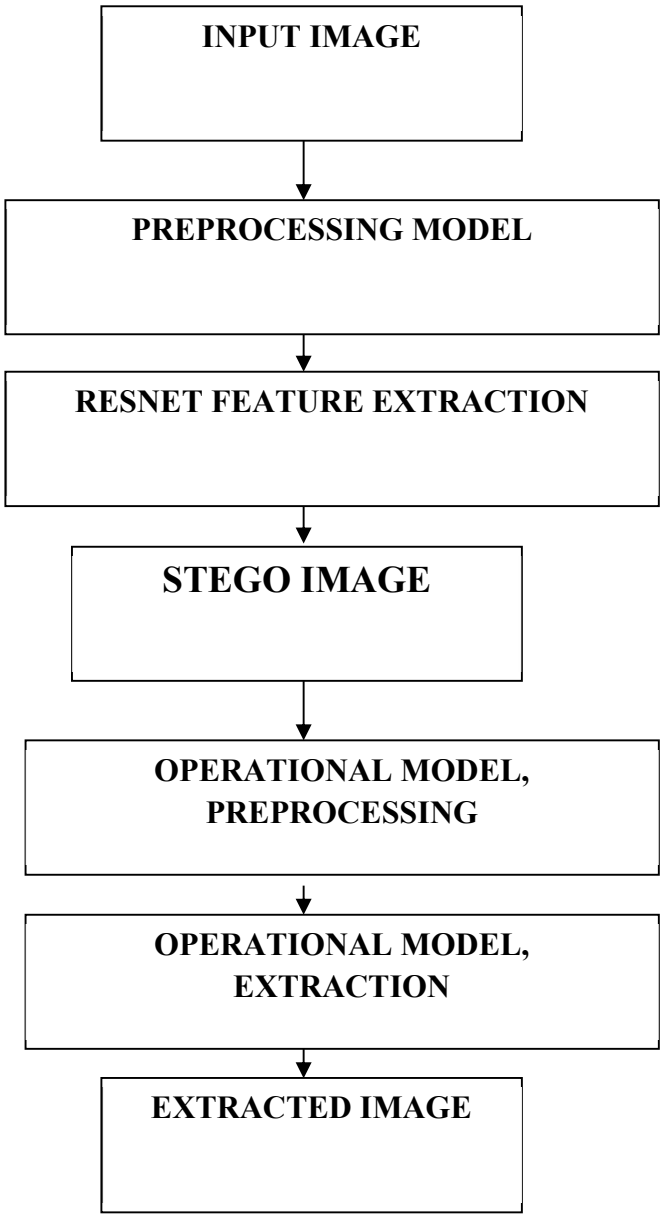


Figure 3.7: High Level Block Diagram

A high-level block diagram in Figure 3.7 showing the data flow across the various phases of the suggested image steganography system is provided. Below is a synopsis of every component:

1. **Input Image:** Identifies the original image in which a hidden message is to be concealed.
2. **Preprocessing Model:** First processing of the input image using methods including blurring, scaling, and normalization.

3. **ResNet Feature Extraction:** Using the ResNet architecture, robust features are extracted from the image that has been preprocessed.
4. **Operational Model (Embedding):** Incorporates the secret message into the ResNet model's features.
5. **Stepgo Image:** i) The image that is produced once the secret message has been embedded.
6. **Model of Operations (Extraction):** Utilizes the same operational model design to retrieve the secret message from the stego image.
7. **Extracted Secret Image**: i) The completed result that shows the secret image that was successfully extracted.

Figure 3.7 gives a general idea of how data moves through the many phases of the system, from initial processing to extraction and embedding, which leads to the extraction of the concealed message.

There are two primary parts to the operational model:

1. **Residual Block Connections (ResNet-like):**
   I) These relationships play a critical role in solving the vanishing gradient issue that deep Convolutional Neural Networks (CNNs) face. Important features can be reused thanks to residual blocks. Element-wise addition of features from various levels improves information flow.
   II) The operational paradigm of the suggested system contains three residual block connections: Batch Normalization (BN) layers, Leaky-ReLU layers, and two convolution transpose operations. These layers replicate an image from a feature map, basically acting as a decoder in an autoencoder.

2. **Convolutional Layers:**
   I) Convolutional layers are used in the operational model for both the embedding and extraction stages. These layers are in charge of producing feature maps and handling various phases of data processing.

II) Each phase of the suggested system contains two convolutional layers (embedding and extraction). There are 32 filters in each phase's first convolutional layer, 64 filters in the embedding phase, and 128 filters in the extraction phase of the second convolutional layer.

III) For both phases, every convolutional layer has a 3x3 kernel, a stride of 2, and a padding size of 1.

3. Activation Functions and Batch Normalization:

I) In both the convolutional layers and the ResNet-like connections, ReLU activation functions are used after each convolutional layer. ReLU adds non-linearity, which facilitates the learning of picture details.

II) To improve training efficiency and expedite network training, Batch Normalization (BN) layers are applied following all convolutional layers, with the exception of the first and last.

4. Stride, Padding, and Kernel Sizes:

I) The convolutional layers' kernel sizes are 3x3. For every convolutional layer, the padding size is 1, and the stride is set to 2. The size and caliber of features that the convolutional layers extract are managed by these parameters.

To sum up, the operational model is built with connections akin to ResNet to tackle problems such as the vanishing gradient problem. For efficient feature extraction and image production throughout both the embedding and extraction stages, it makes use of convolutional layers, activation functions, and batch normalization.

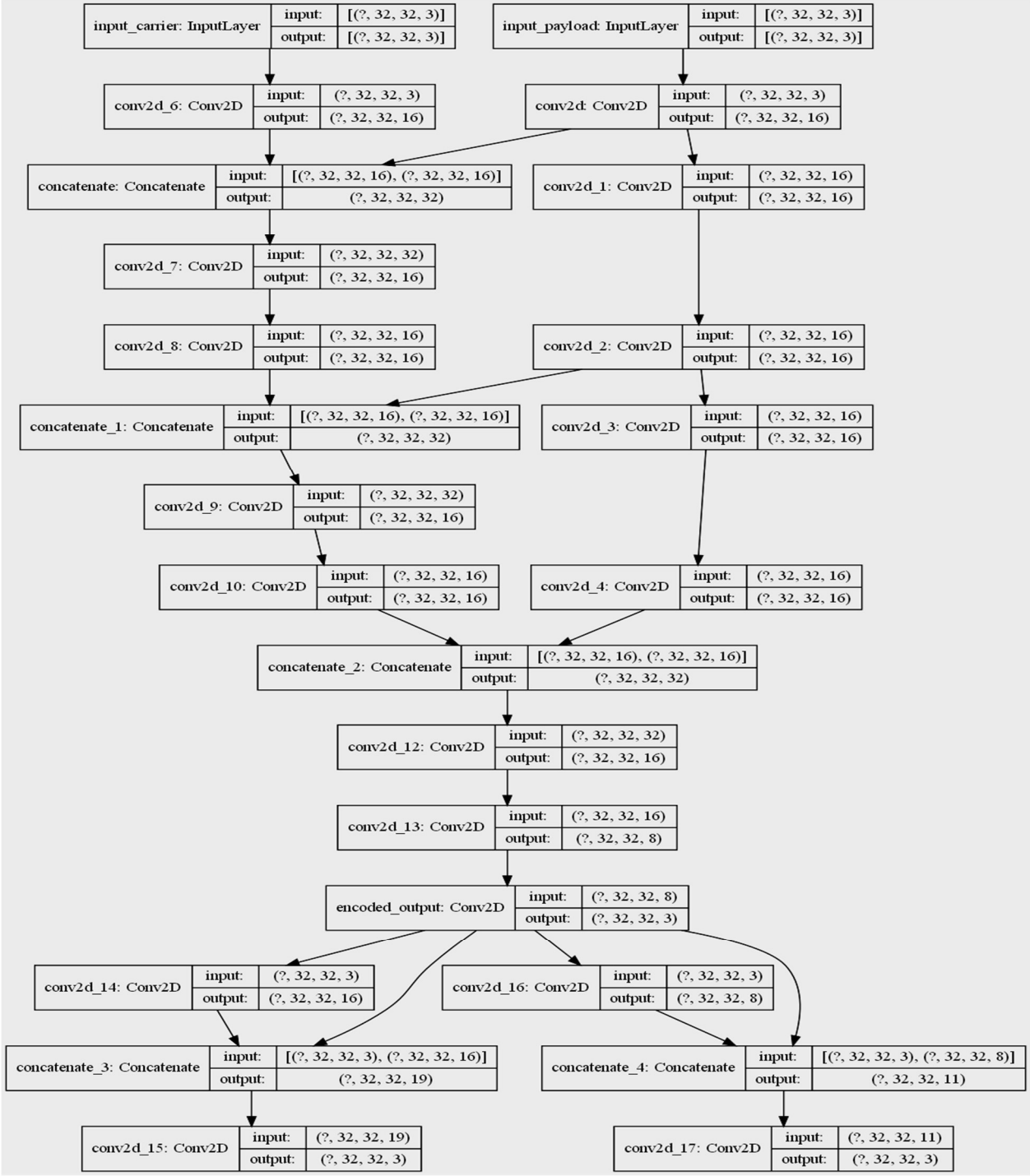


Figure 3.8: Model Summary

The model summary shown in Figure 3.8 provides details on the layers, output shapes, and parameters of the steganography model. Here is an excerpt from the model summary:

Table 3.1: Model Summary

Layer(type)	Output Shape	Param #	Connected to
input_carriertype(Input Layer)	None[(32, 32, 3)]	0	
input_payloadtype(Input Layer)	None[(32, 32, 3)]	0	
Conv2D_6	None[(32, 32, 16)]	448	input_carrier[0][0]
Conv2D_17	None[(32, 32, 3)]	300	Concatenate_4[0][0]

Total Param = 26,243

Trainable Params = 26,243

Non- trainable Params = 0

The encoded steganographic image and two decoded outputs (recovered cover image and secret information) are the three major outputs of the model, which has many convolutional layers.

Loss Function and Optimization

Loss Function

In order to train the model, two loss functions are defined:

1. {Branched loss function}: This loss function takes into account the differences between the encoder output and the host (cover image), the payload (secret information) and the decoded payload, and the host and the decoder host output. To determine the total loss, it combines these variations with weights (alpha, beta, and gamma) that can be customized.

2. Loss function: This streamlined loss function just takes into account variations between the encoder output and the host, as well as between the payload and the decoded payload.

Optimization

The Adam optimizer is used to train the model at a learning rate of 0.0001. This optimizer adjusts the model's weights in order to minimize the loss function during training.

3.3.10 Training Process

There are 250 training epochs in all for the model. The steps are as follows for each era:

1. Data shuffle: To guarantee unpredictability in each epoch, the training and test datasets are shuffled.
2. Normalization: To guarantee that pixel values in cover photos and classified information are constant, data normalization is utilized.
3. Training and Testing: The model is trained on the training dataset, and its performance is assessed on the test dataset. Training parameters are determined, such as loss, structural similarity index (SSIM), and peak signal-to-noise ratio (PSNR).
4. Metrics Collection: During the training and testing phases, data on metrics including loss, PSNR, and SSIM is collected and stored for examination.

3.3.10.1 Model Weights Saving

Because they retain the patterns that are learned during training, the model weights are a crucial part of the steganography model. Preserving the model weights is essential for subsequent applications, such as retraining and inference. The file weights contain the model weights that are saved.h5use the code found below:steganography_model.weights_savingsthe string.

"./model_weights/weights.h5"

The model's weights are preserved for later usage thanks to this little piece of code.

3.3.10.2 Training Results Storage

It is essential to keep an eye on the training process in order to assess model performance and identify areas for improvement. To store training and testing metrics independently, two Data Frames are created, named train_df and test_df. For the host (cover picture) and payload (secret data), these parameters include loss, PSNR (Peak Signal-to-Noise Ratio), and SSIM (Structural Similarity Index).

train_df = pd.DataFrame(train_metrics, columns=['Epoch', 'Train Loss', 'TrainH PSNR', 'TrainP PSNR', 'TrainH SSIM', 'TrainP SSIM'])

test_df = pd.DataFrame(test_metrics, columns=['Epoch', 'Test Loss', 'TestH PSNR', 'TestP PSNR', 'TestH SSIM', 'TestP SSIM'])

The training and testing metrics are organized into DataFrames and are later saved to an Excel file for further analysis. The Excel file, named **training_results.xlsx**, is created using the **pd.ExcelWriter**contextmanager.

Testing Process with Normalization Modification

Data Preparation for Testing

For testing purposes, random cover images and payload images from the test dataset are selected. These images are normalized using a modified normalization function **test_normalize**. This normalization function ensures that pixel values are within a specific range, facilitating consistent testing.

Testing Metrics

The steganography model is evaluated using the test dataset, and various metrics are computed, including:

i. Test Loss: A measure of the overall loss on the test dataset.

ii. PSNR-H (Peak Signal-to-Noise Ratio for Host): A metric indicating the quality of the recovered cover images.
iii. PSNR-P (Peak Signal-to-Noise Ratio for Payload): A metric indicating the quality of the extracted secret information.

3.3.11 Results and Conclusion

In order to efficiently embed and extract secret information from cover images while preserving good image quality, the steganography model must be optimized during the training phase. Metrics like loss, PSNR, and SSIM that are gathered offer information on how well the model performs over time.

3.3.12 System Use case

A CNN (Convolutional Neural Network) use case diagram for an image steganography system shows the main interactions and functionalities of the system from a high-level viewpoint.

1. Actor: The User

2**. Use Cases:**

a. **Encode Image:** The user initiates the process by selecting an image they want to hide a secret image within. The system takes this cover image and a secret image as input.

b. Decoding Image: The user initiates this process when they want to extract the secret image from a stego image. The system takes the stego image as input.

3. Interactions:

Encode picture: The cover picture and secret image are provided by the user in order to communicate with the system. The photos are pre-processed by the system through a process of normalization and format conversion. A CNN-based preprocessing model with two convolutional layers is used by the system. This model reduces redundancy by extracting important information from both the cover and secret photos. Using a predetermined method, the features that were retrieved from the cover and hidden photos are combined. The hidden image is successfully embedded into the cover image by the system through the use of an Operational

Model, which also makes use of CNN architecture. The user is presented with the created stego image. Additionally, the system might include a number of steganography level adjustment features, like alpha ($\boldsymbol{\alpha}$) value management.

Decode Image: The stego image is provided by the user in order to communicate with the system. The stego image is pre-processed by the system so that it is ready for extraction. To extract the secret image from the stego image, the system makes use of the same Operational Model that was used during the encoding process in conjunction with a ResNet connection. The user is shown the secret image that was extracted.

Use case Diagram

Brief Explanation of Users and Interactions

Users:

1. **End User**: The primary user interacting with the system. This user provides input images, views the output, and evaluates the results.
2. **System Administrator**: Manages and oversees the system operations, ensuring the system run smoothly and securely.

Interactions:

1. **Input Secret Image**: The user provides the image that contains the information to be hidden.
2. **Input Cover Image**: The user provides the cover image in which the secret image will be embedded.
3. **Encode**: The system processes the input images using the integrated CNN and ResNet to embed the secret image into the cover image, producing the stego image.
4. **View Stego Image**: The user views the stego image, which appears similar to the original cover image but contains the hidden secret image.
5. **Image Extracted by CNN ResNet**: The system uses CNN and ResNet to extract the hidden secret image from the stego image.
6. **Decode Image**: The system decodes the extracted data to retrieve the secret image.

7. **View Original Image**: The user views the original secret image to verify the accuracy and effectiveness of the embedding and extraction processes.

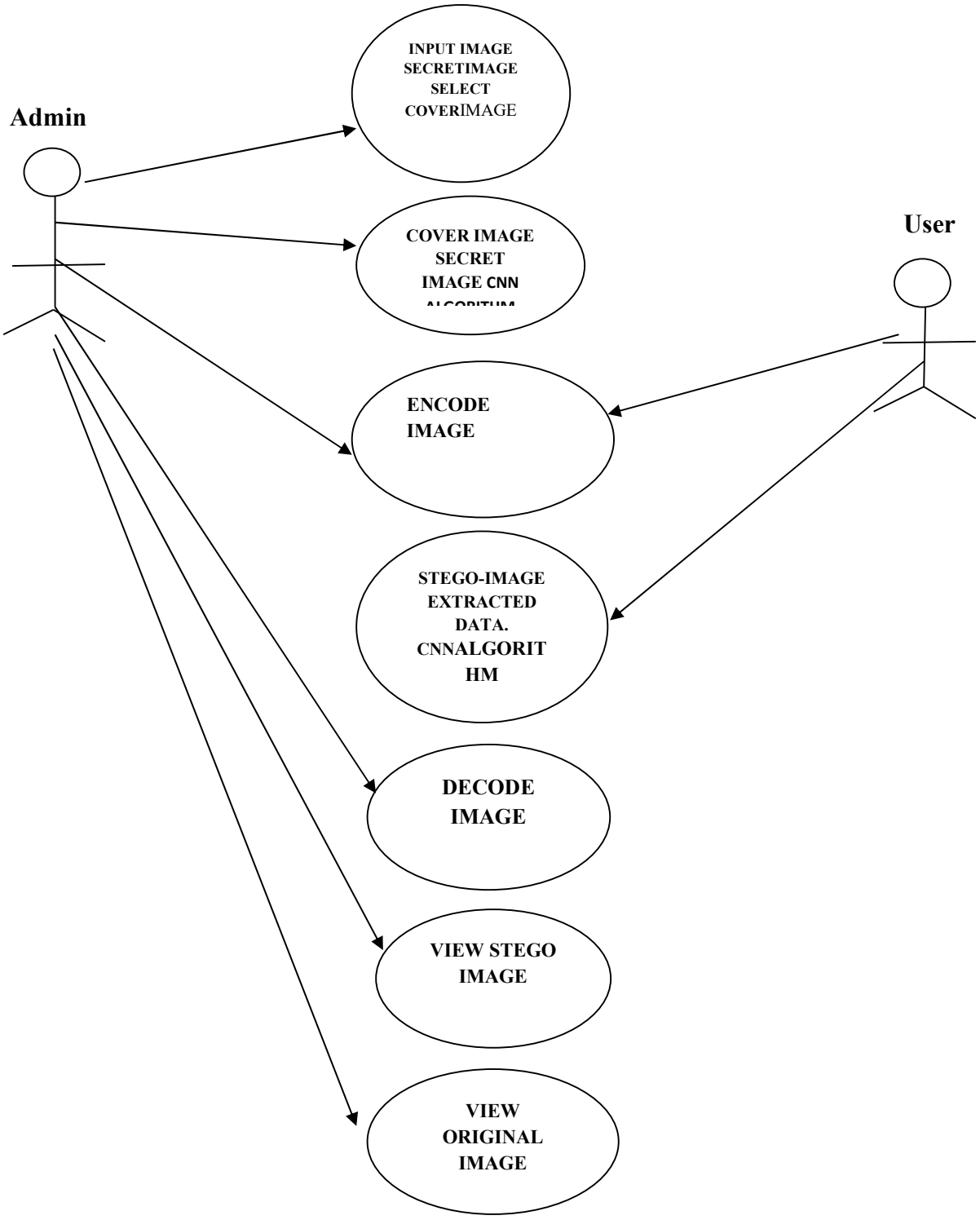


Figure 3.9: Use Case Diagram of the Proposed System

An overview of the CNN-based picture steganography system's operation is shown in this use case diagram (Figure 3.9). In order to balance security and image quality, it illustrates the essential interactions and results for users who want to conceal and retrieve hidden images from cover photographs.

3.4 Summary of the experimental set up

Preprocessing Steps: The dataset images were subjected to preprocessing techniques such as augmentation, normalization, and resizing before to model training. Resizing made guaranteed that all the photographs had the same dimensions, and normalization standardized the pixel values so that training would be easier. To improve model generalization and increase the diversity of training data, augmentation techniques including rotation and flipping were used. Hyperparameters and Experimental Design: Based on this, the CNN and ResNet model training hyperparameters were set up. In order to minimize overfitting and maximize model performance, many parameters were adjusted, including learning rate, batch size, and number of epochs. To ensure an objective assessment of the models' performance, the datasets were divided into training, validation, and testing sets as part of the experimental design.

Overall, the proposed CNN and ResNet-based picture steganography system was trained, tested, and evaluated using a strong framework made possible by the experimental setup, allowing for a thorough examination of the system's efficacy and efficiency.

3.4.1 Software Development Tools

In the course of the project, several software development tools were utilized for training, designing, and implementing the steganography algorithms and deep learning models. These tools provided essential functionalities for building and deploying robust and efficient systems.

1. **Python:** Python served as the primary programming language throughout the project and was instrumental in implementing steganography algorithms and developing deep learning models using Convolutional Neural Networks (CNNs), Residual Networks (RSNets), and Autoencoders (AEs).

2. **TensorFlow**: an open-source machine learning library developed by Google, played a crucial role in building, training, and deploying deep learning models. Leveraging TensorFlow's extensive set of pre-built functions and models; we were able to construct sophisticated neural networks

3. **Keras:** Built on top of TensorFlow, Keras enabled us to rapidly prototype, train, and evaluate various deep learning models for steganography, ensuring efficient development workflows.

4. **PyCharm**: provided comprehensive support for writing, testing, and debugging Python code.

3.4.2 Hardware Specifications

Here are the hardware specifications utilized: Laptop and desktop computer with keyboard and mouse**,** CPU: Intel Core i5 7 processor**,** GPU: Nvidia GeForce RTX 2080, **RAM**: 8GB**,** Storage: SSD with 512GB of storage space**,** Operating System: Windows 10 and Linux**,** UPS Unit for power backup.

Generally, a system for image steganography using deep learning algorithms such as CNN, autoencoders, and ResNet would require a powerful CPU and GPU to handle the heavy computational load of training and testing the models.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Results

This chapter presents the results of the system evaluation, results from the different modules and interfaces, a table with the usability test results, and scatter plots illustrating the dependencies between the home and the system.

4.2 Interfaces and Modules

The home screen (Figure 4.1) serves as the user's starting point and gives an overview of the application, whereas the screen or page shows as soon as the user initializes the system.

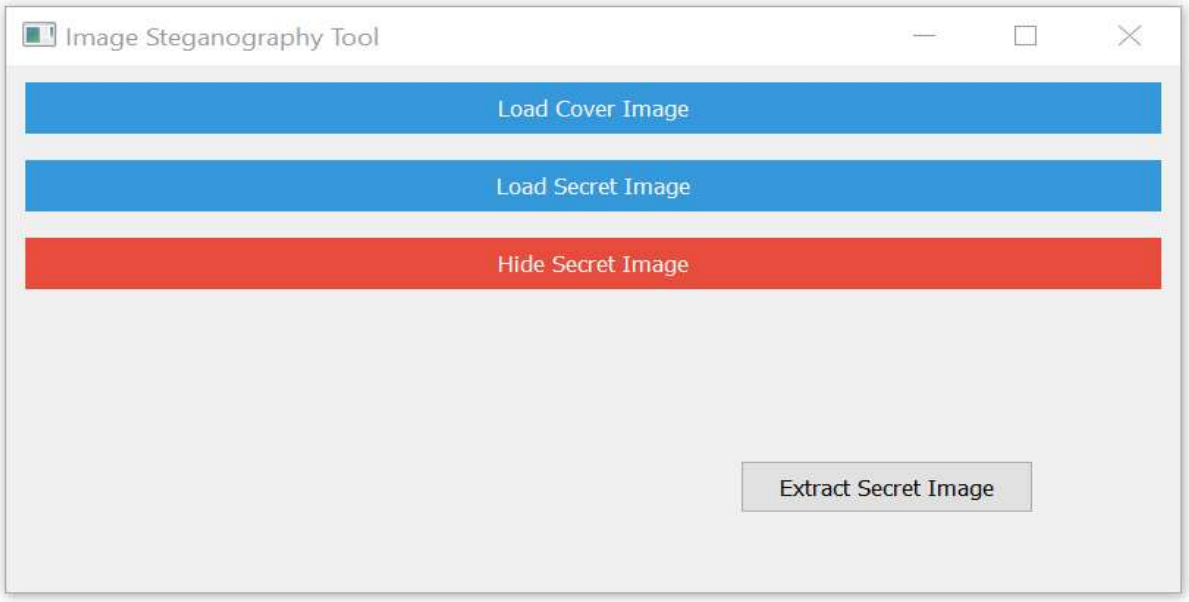


Figure 4.1: Software interface (home screen)

1. Load Cover Image Button:

Function: The user can import or load a cover photo from their smart phone by pressing this button. The public or viewable image that will conceal sensitive information is called the cover image. Use: Selecting the cover image to hide the hidden data involves clicking this button, which displays a file explorer or dialog box (Figure 4.2).

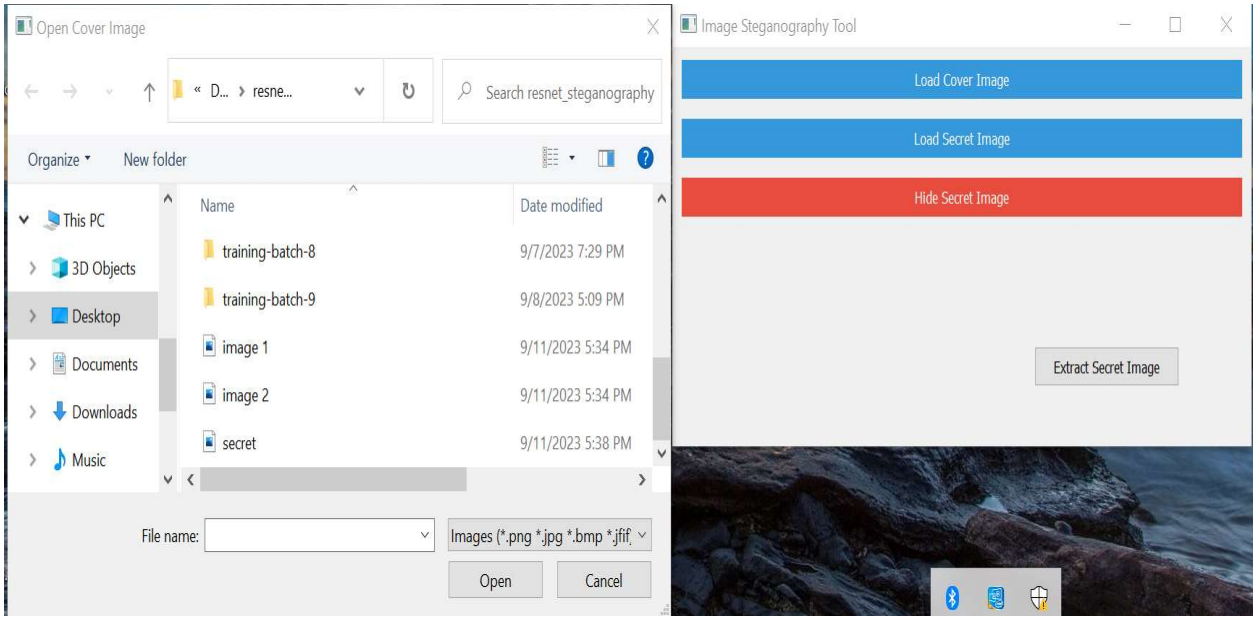


Figure 4.2: Uploading cover image

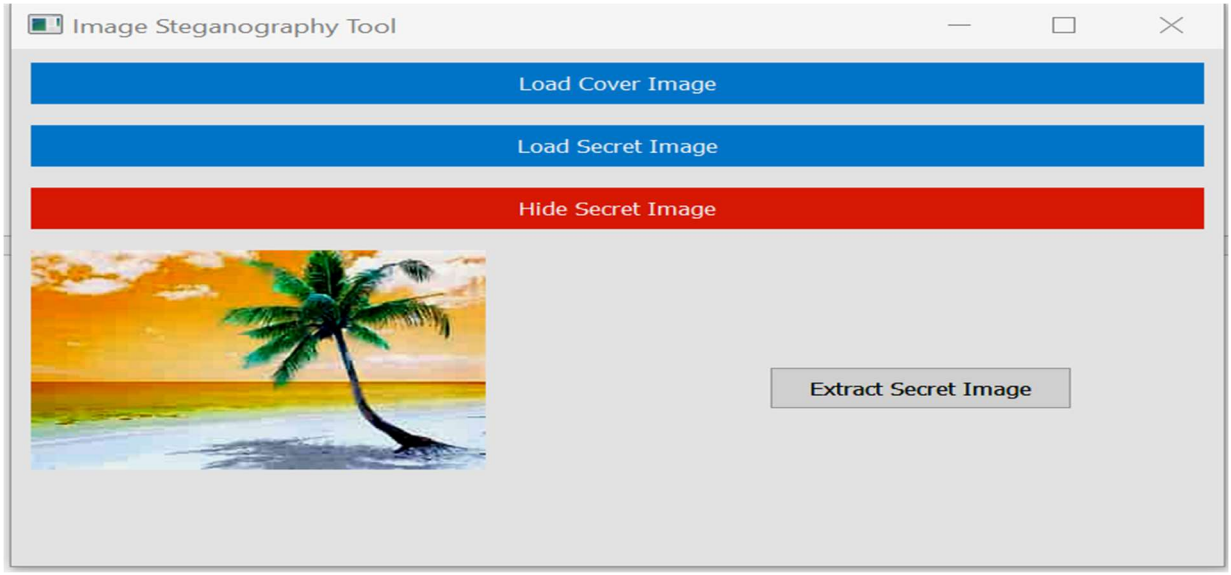


Figure 4.3: Cover image uploaded.

1. **Load Secret Image Button**:

Illustrated in Figure 4.3 is the Cover Image Upload Page where the user can load or choose a secret image to be hidden behind the cover image by using the button. Selecting this button causes a file explorer or dialog box to popup, giving the user the option to hide a secret image inside the cover image.

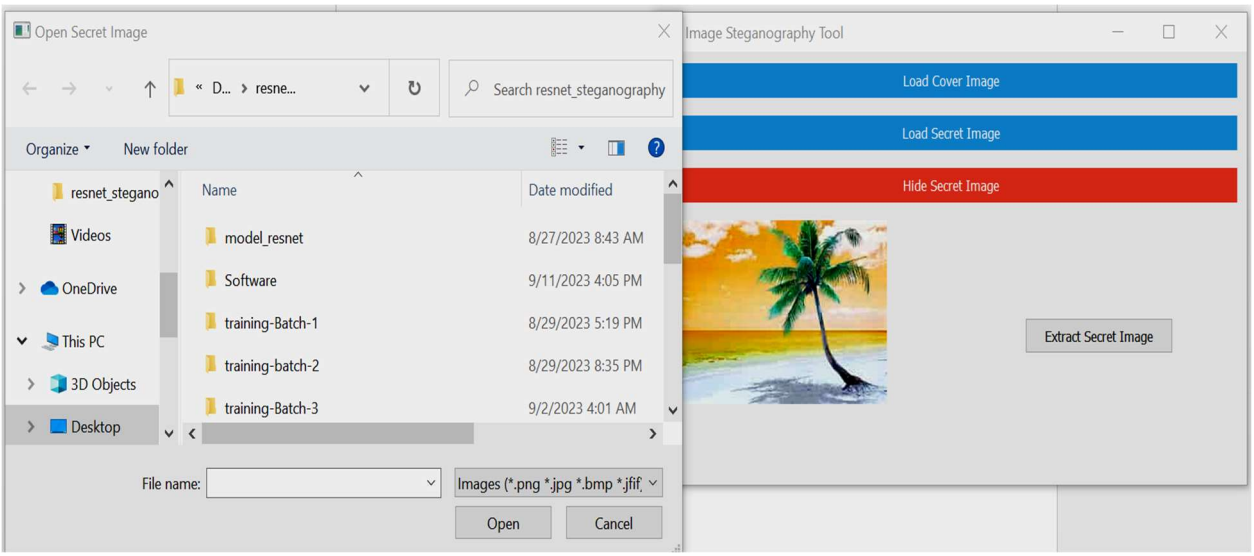


Figure 4.4: Uploading secret image.

Users can select a secret image (Figure 4.4) containing the information they want to keep hidden by using the "load secret image" function. Users can submit their own secret image or choose from a collection of photographs, just like when choosing a cover image. The steganographic procedure keeps this image hidden inside the cover image. For reference, the secret image that was chosen is also shown on the interface (Figure 4.5).

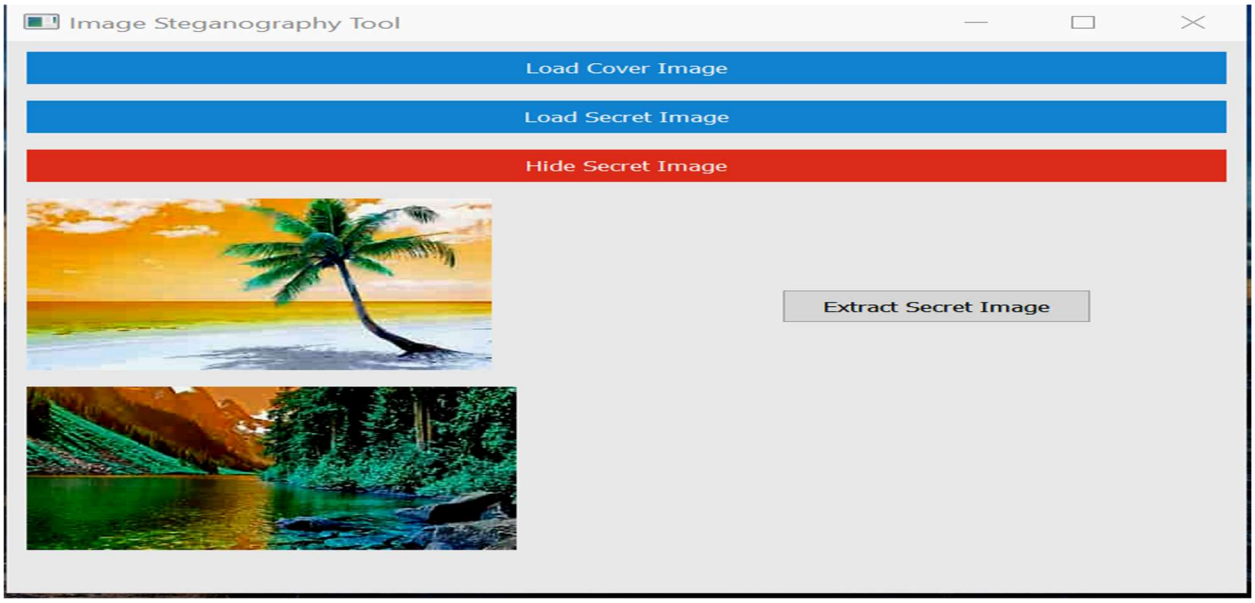


Figure 4.5: Secret image uploaded.

2. Hide Secret Image Button:

Selecting the "hide secret image" option starts the process of integrating the secret data into the cover image once the cover and secret images have been loaded. In order to protect the cover image's aesthetic integrity while hiding the secret image within its pixels, this stage makes use of steganographic techniques. Users receive proof that their concealing efforts were effective once the hiding process is ended, and they can download the updated cover image that now has the hidden information contained in it.

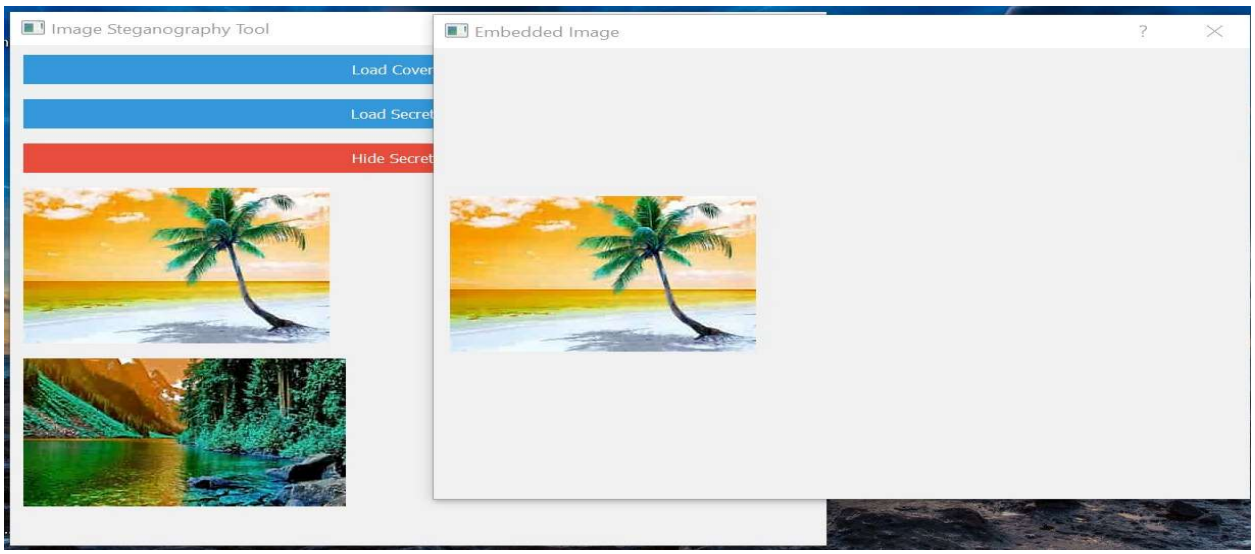


Figure 4.6: Stego Image displayed after Embedding

The Stego picture produced by the image steganographic technology is displayed in Figure 4.6. Using our cutting-edge steganographic techniques, we were able to successfully conceal the secret image inside a cover image. The outcome is captured in the screenshot. The system's integrated image display page is a turning point in our creative picture steganography methodology. Using ResNet-inspired structures and Convolutional Autoencoders, our system smoothly embeds a secret image into a cover image.

2. Extract Secret Image Button:

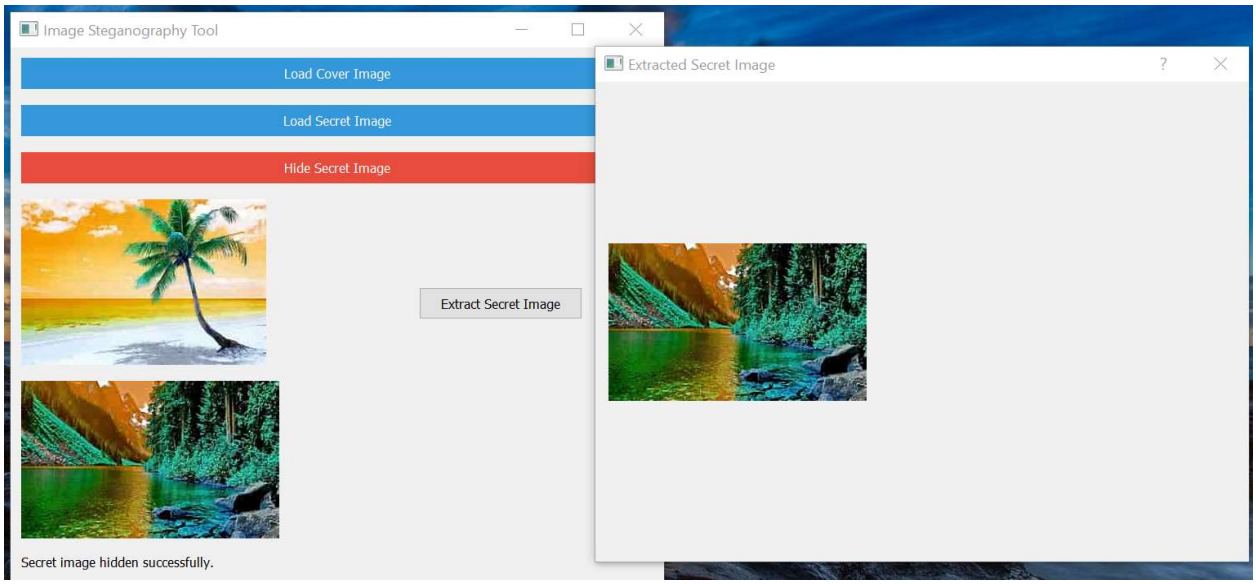


Figure 4.7: Secret image extracted from cover image successfully

Extract Secret Image: Using this button, you can take the concealed picture out of a Stego image. An easy-to-use method of interacting with the image steganography technology is to reverse the process of hiding the SUI home screen. The cover image can be loaded along with the secret image, which can then be extracted when needed, as shown in Figure 4.7. The secret image can be hidden inside the cover image. When the user wishes to retrieve the concealed data, these buttons help with the full process of hiding and retrieving it from the secret image.

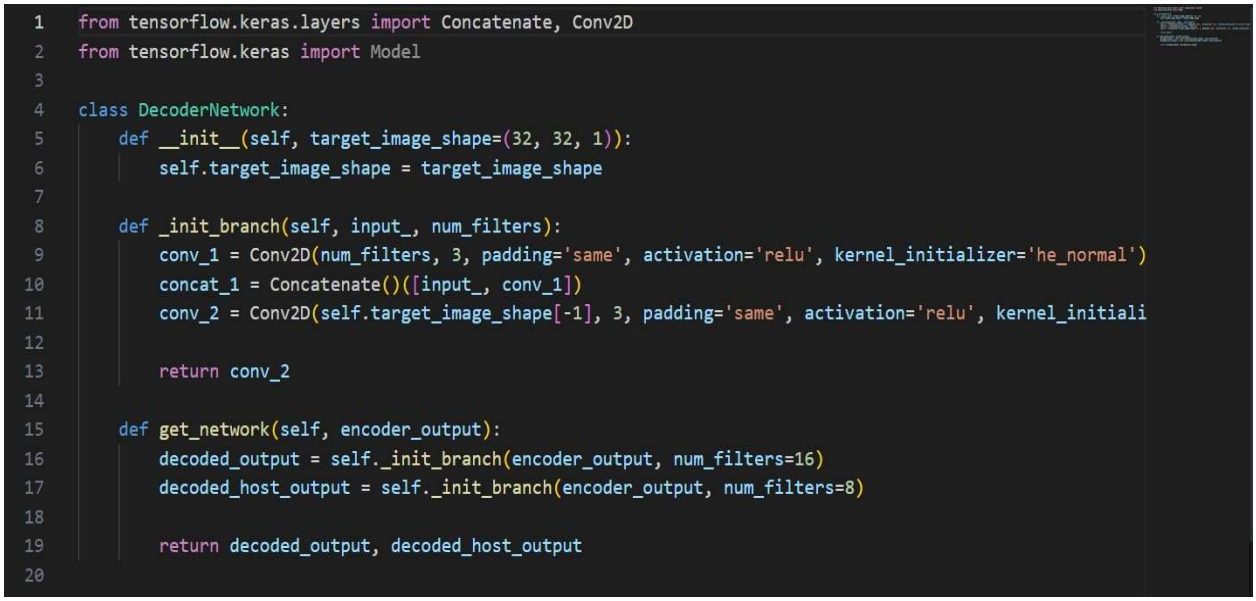
Use: Pressing this button initiates the stego picture extraction procedure, exposing the hidden secret image.

In conclusion, the home screen of the software interface provides a simple means of interacting with the image steganography technology.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense, Flatten, Conv2D, Concatenate, Input

class EncoderNetwork:
    def __init__(self, carrier_shape=(32, 32, 3), payload_shape=(32, 32, 1)):

#         super(EncoderModel, self).__init__()
        self.carrier_shape = carrier_shape
        self.payload_shape = payload_shape

    def _init_branch_payload(self, payload):

        self.branch_payload_conv_1 = Conv2D(16, 3, padding='same', activation='relu', kernel_initializer=
        self.branch_payload_conv_2 = Conv2D(16, 3, padding='same', activation='relu', kernel_initializer=
        self.branch_payload_conv_3 = Conv2D(16, 3, padding='same', activation='relu', kernel_initializer=
        self.branch_payload_conv_4 = Conv2D(16, 3, padding='same', activation='relu', kernel_initializer=
        self.branch_payload_conv_5 = Conv2D(16, 3, padding='same', activation='relu', kernel_initializer=
        self.branch_payload_conv_6 = Conv2D(16, 3, padding='same', activation='relu', kernel_initializer=

        self.payload_tensors = [self.branch_payload_conv_1, self.branch_payload_conv_2,
                                self.branch_payload_conv_3, self.branch_payload_conv_4,
                                self.branch_payload_conv_5, self.branch_payload_conv_6]

    def _init_branch_carrier(self, carrier):

        self.branch_carrier_conv_1 = Conv2D(16, 3, padding='same', activation='relu', kernel_initializer='
        self.branch_carrier_concat_1 = Concatenate()([self.branch_carrier_conv_1, self.branch_payload_con
```

Figure 4.8: Snippet from the encode module

The encode module's primary purpose, which is inserting secret data into the cover image, is demonstrated by the code excerpt from Figure 4.8.



```python
from tensorflow.keras.layers import Concatenate, Conv2D
from tensorflow.keras import Model

class DecoderNetwork:
    def __init__(self, target_image_shape=(32, 32, 1)):
        self.target_image_shape = target_image_shape

    def _init_branch(self, input_, num_filters):
        conv_1 = Conv2D(num_filters, 3, padding='same', activation='relu', kernel_initializer='he_normal')
        concat_1 = Concatenate()([input_, conv_1])
        conv_2 = Conv2D(self.target_image_shape[-1], 3, padding='same', activation='relu', kernel_initiali

        return conv_2

    def get_network(self, encoder_output):
        decoded_output = self._init_branch(encoder_output, num_filters=16)
        decoded_host_output = self._init_branch(encoder_output, num_filters=8)

        return decoded_output, decoded_host_output
```

Figure 4.9: Snippet from the decoder module

The process of extracting the secret data into the cover image is the essential operation of the decode module, as demonstrated by the code snippet from Figure4.9.

4.3 Results of Usability Test

Table 4.1 describes the various activities done during testing of each module, the processes involved as well as the various outputs and results.

Table 4.1 Results of Usability Test

SN	**Activity**	**Process**	**Output/Result**
	Module: Encode		
1	User uploads cover image	User selects a cover image, uploads a secret image and initiates the encoding	Stegonagraphic image (I_stego)
	Module: Decode		
2	User uploads a steganographic image	User selects a steganographic image and initiates a decoding process	Extracted image
3	Module: Steganalysis User uploads steganographic images for analysis	User selects a steganographic image and initiates steganalysis	Detection rate, analysis details

4.4 System Evaluation Result

Nine batches totaling 250 epochs were used for the training process. This training was primarily concerned with assessing the model's performance with respect to image quality, distortion, and embedding efficiency.

Training Batches Overview

The training process was divided into nine batches, each with 250 epochs. Below is a summary of the key findings for each batch:

Batch 1 (See Figure 4.10)

- PSNR-H (Cover Pictures): 19.15

- SSIM-H (Cover Pictures): 0.74

- PSNR-P (Secret Image): 20.85

- SSIM-P (Secret Image): 0.83

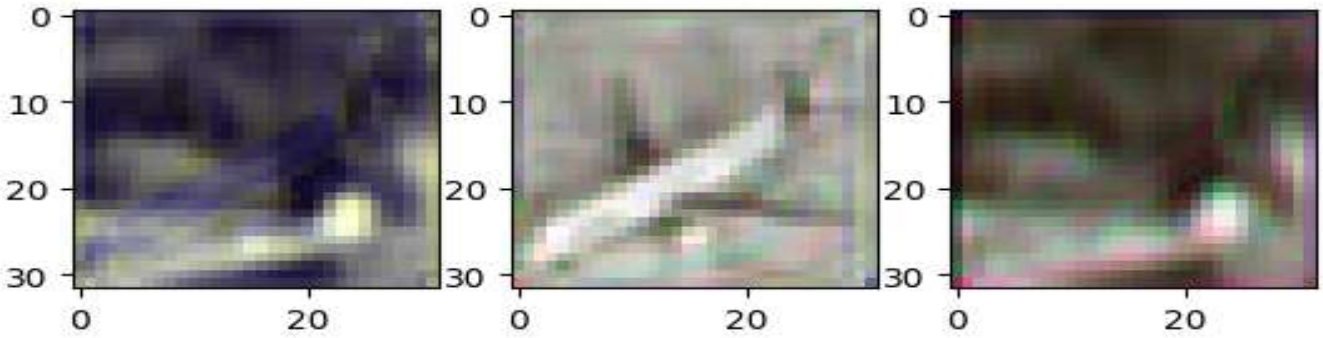


Figure 4.10: Batch 1 – Initial Training – High Distortion

We noticed certain anomalies in the cover photos, secret images, and embedding procedure in the initial batch. The quality of the recovered hidden image had increased somewhat.

Batch 2 (See Figure 4.11)

- PSNR-H (Cover Pictures): 20.13

- SSIM-H (Cover Pictures): 0.80

- PSNR-P (Secret Image): 23.30

- SSIM-P (Secret Image): 0.87

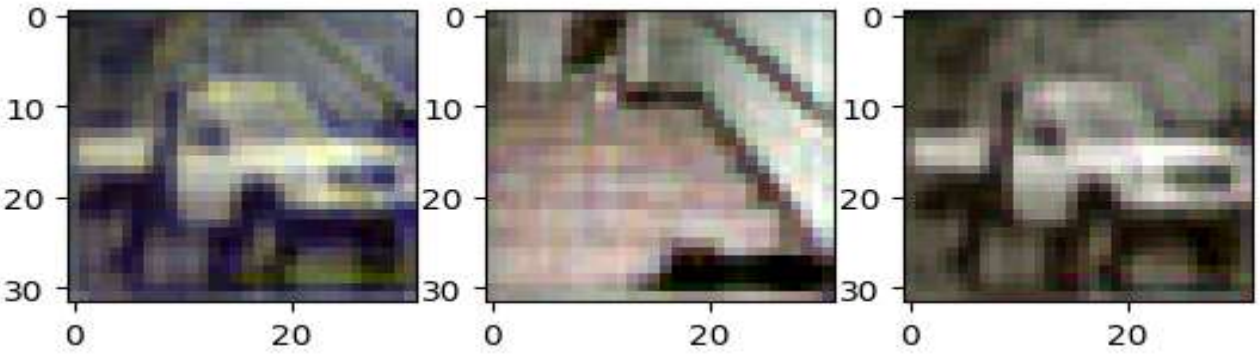


Figure 4.11: Batch 2 – Gradual Improvement in Embedding

In comparison to the first batch, the second batch showed a small improvement in image quality. Both the cover photo and the secret image had higher PSNRs, which suggests less distortion.

Batch 3 (See Figure 4.12)

- PSNR-H (Cover Pictures): 20.32

- SSIM-H (Cover Pictures): 0.81

- PSNR-P (Secret Image): 24.73

- SSIM-P (Secret Image): 0.90

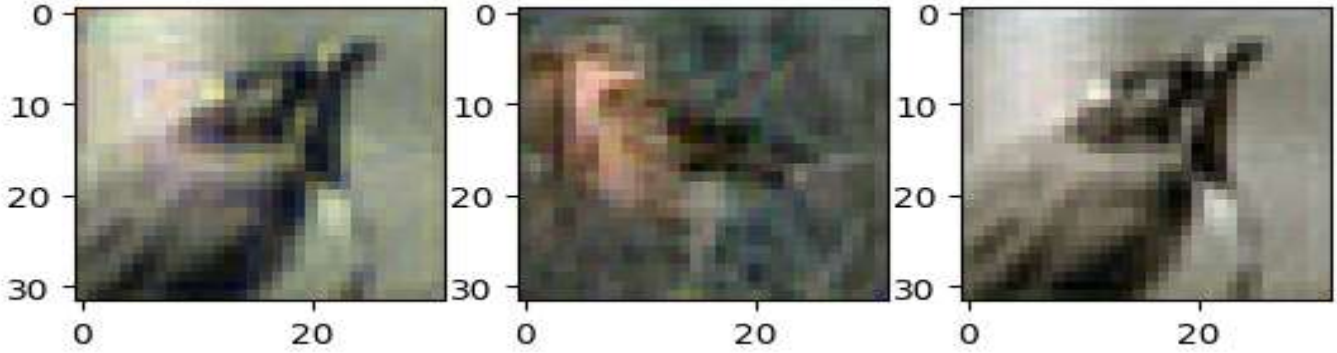


Figure 4.12: Batch 3 – Enhanced Feature Learning

Batch 3 exhibited further enhancement in image quality. Distortions in both cover pictures and secret images were reduced, and the extracted secret image demonstrated a notable increase in quality.

Batch 4 (See Figure 4.13)

- PSNR-H (Cover Pictures): 20.61

- SSIM-H (Cover Pictures): 0.83

- PSNR-P (Secret Image): 25.43

- SSIM-P (Secret Image): 0.91

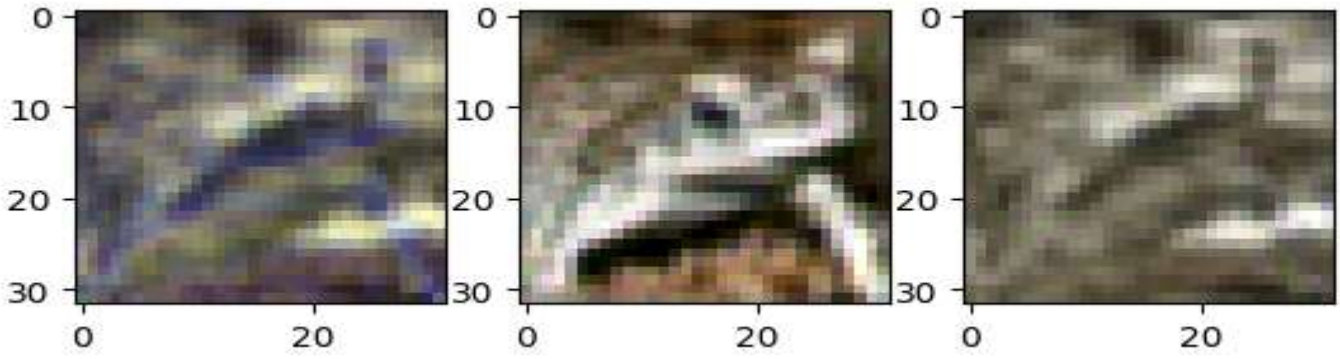


Figure 4.13: Batch 4 – Balanced Quality Improvement

The quality of the images was further enhanced in the fourth batch. The quality of the extracted hidden image significantly improved, and distortions diminished.

Batch 5 (See Figure 4.14)

- PSNR-H (Cover Pictures): 20.91

- SSIM-H (Cover Pictures): 0.93

- PSNR-P (Secret Image): 25.93

- SSIM-P (Secret Image): 0.92

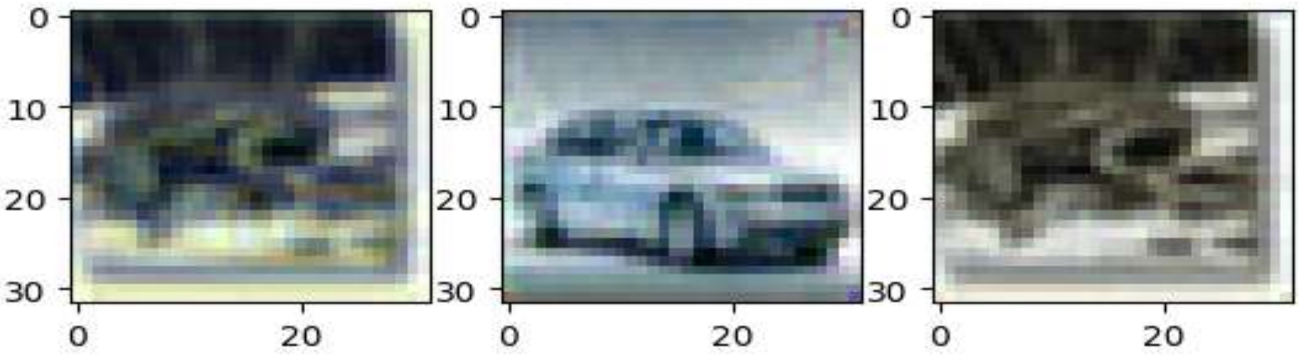


Figure 4.14: Batch 5 – Advanced Image Recovery

Both the cover photo and the secret image in the sixth batch have slight distortions. Nonetheless, there was a noticeable improvement in the quality of the hidden image recovered and the embedding procedure.

Batch 6 (See Figure 4.15)

- PSNR-H (Cover Pictures): 26.09

- SSIM-H (Cover Pictures): 0.92

- PSNR-P (Secret Image): 21.08

- SSIM-P (Secret Image): 0.85

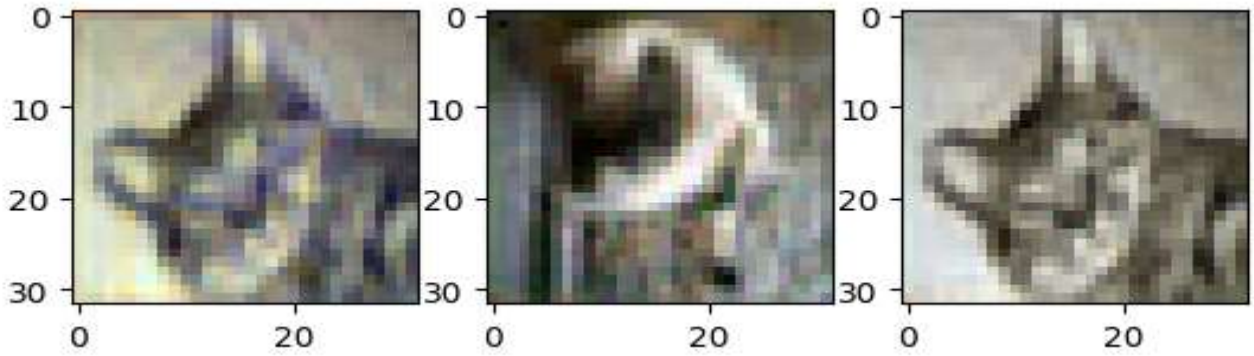


Figure 4.15: Batch 6 – Refinement with Minimal Deformities

The cover photos and hidden images from the sixth batch showed minor deformities. On the other hand, the hidden image that was extracted showed better image quality.

Batch 7 (See Figure 4.16)

- PSNR-H (Cover Pictures): 22.08

- SSIM-H (Cover Pictures): 0.85

- PSNR-P (Secret Image): 27.37

- SSIM-P (Secret Image): 0.92

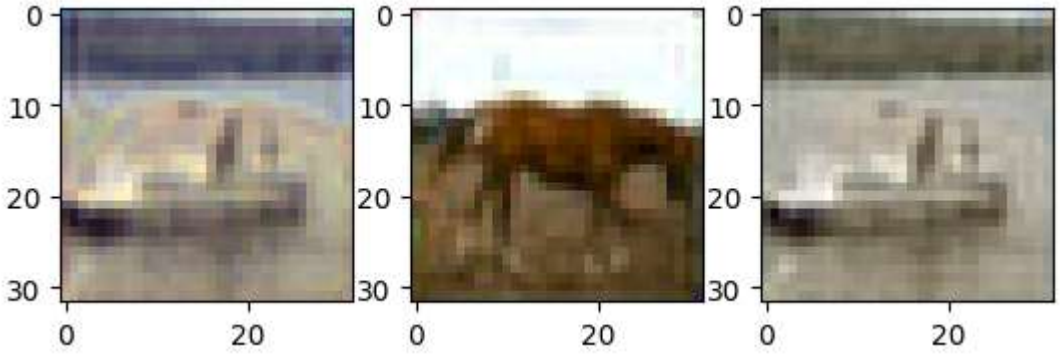


Figure 4.16: Batch 7 – Increased Robustness

Cover photos and secret images continued to be distorted in the seventh batch, while the quality of the hidden image that was retrieved greatly improved.

Batch 8 (See Figure 4.17)

- PSNR-H (Cover Pictures): 23.39

- SSIM-H (Cover Pictures): 0.86

- PSNR-P (Secret Image): 27.94

- SSIM-P (Secret Image): 0.92

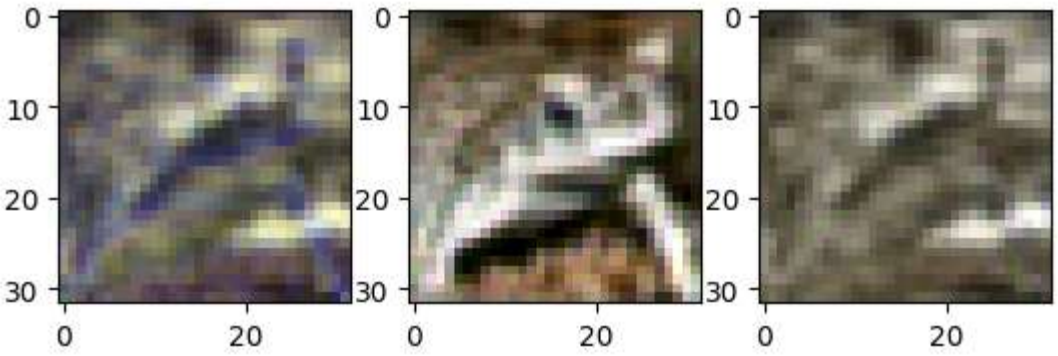


Figure 4.17: Batch 8 – Near Optimal Stego Embedding

The cover photos and hidden images of the ninth batch have minor deformities.

The quality of the secret image that was extracted, though, kept becoming better.

Batch 9 (See Figure 4.18)

- PSNR-H (Cover Pictures): 26.08

- SSIM-H (Cover Pictures): 0.87

- PSNR-P (Secret Image): 31.85

- SSIM-P (Secret Image): 0.92

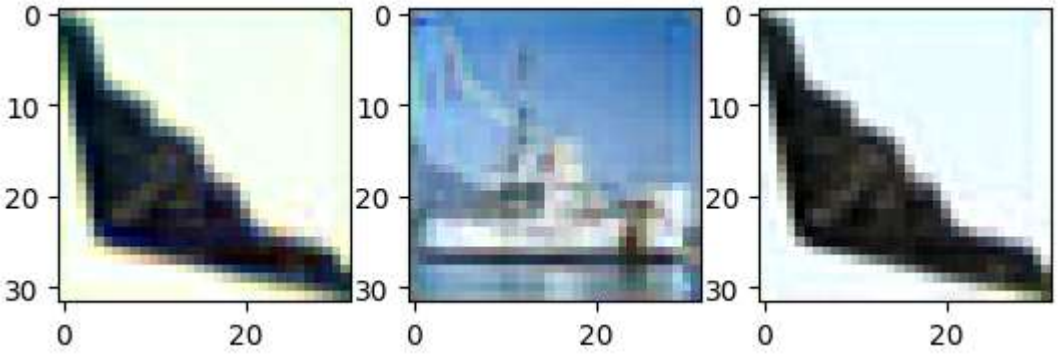


Figure 4.18: Batch 9 – Optimal Quality – Final Model

Very little distortion was seen in cover photos and hidden shots from the ninth and final batch. The hidden image that was recovered showed a notable improvement in image quality.

In conclusion, there has been a noticeable improvement in image quality and distortion reduction after our Image Steganography model was trained using CNN Autoencoders built on the ResNet architecture. The quality of embedded secret photos has consistently improved over the course of the nine training batches.

In summary, the new Image Steganography model is a useful tool for secure information transmission and storage since it has the potential to hide and retrieve concealed information from images while preserving image quality. Future editions of the can potentially achieve even higher performance by exploring further modifications and optimizations.

For the suggested approach, the PSNR and SSIM values of a single cipher10 dataset were determined.

The average PSNR and SSIM values of the suggested scheme, which are chosen at random from datasets and the outcomes of alternative steganography techniques, are displayed in Table 4.2.

Table 4.2: Comparing Different Steganography Methods' PSNR and SSIM Values

Methods	**First batch**		**Last Batch**	
	Peak signal to noise Ratio (PSNR)	**Structural Similarity Index Measure (SSIM)**	**Peak signal to noise Ratio (PSNR)**	**Structural Similarity Index Measure (SSIM)**
Traditional Methods	34.159	0.9549	39.254	|| 0.9615
Deep Steganography Methods (COCO Dataset)	($\boldsymbol{\alpha}$ = 0.3) | 39.561	0.9810	40.313	0.9920

Deep Steganography (Proposed Method) Cifar 10 Dataset (α = 0.5)	27.0988	0.9908	31.7949	0.9897

Calculation with Given Parameters

PSNR (Equation 1):

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{Max}^2}{\text{MSE}} \right)$$

SSIM (Equation 2):

$$\text{SSIM}(i, i') = \left(\frac{2\mu_i\mu_{i'} + C_1}{\mu_i^2 + \mu_{i'}^2 + C_1} \right) \cdot \left(\frac{2\sigma_i\sigma_{i'} + C_2}{\sigma_i^2 + \sigma_{i'}^2 + C_2} \right) \cdot \left(\frac{\sigma_{ii'} + C_3}{\sigma_i\sigma_{i'} + C_3} \right)$$

Calculations in Python

Here is the Python code to calculate PSNR and SSIM:

```python

Import cv2

Import numpy as np

From skimage.metrics import structural_similarity as ssim

Import math


# Load the images

Cover_image_path = '/mnt/data/6C755F93-FF81-4D93-A2C5-04147CF9F870.jpeg'  # Change to the actual path

Stego_image_path = '/mnt/data/6C755F93-FF81-4D93-A2C5-04147CF9F870.jpeg'  # Change to the actual path
```

```
Cover_image = cv2.imread(cover_image_path)

Stego_image = cv2.imread(stego_image_path)

# Ensure images are the same size

Assert cover_image.shape == stego_image.shape, “Images must be of the same dimensions”


# Convert images to grayscale

Cover_gray = cv2.cvtColor(cover_image, cv2.COLOR_BGR2GRAY)

Stego_gray = cv2.cvtColor(stego_image, cv2.COLOR_BGR2GRAY)


# Calculate MSE

Mse = np.mean((cover_gray – stego_gray) ** 2)


# Calculate PSNR

Max_pixel = 255.0

Psnr_value = 10 * math.log10(max_pixel**2 / mse)


# Calculate SSIM

Ssim_value, _ = ssim(cover_gray, stego_gray, full=True)

Print(f”Calculated MSE: {mse}”)

Print(f”Calculated PSNR: {psnr_value} dB”)

Print(f”Calculated SSIM: {ssim_value}”)
```

The PSNR of 31.7949 dB and SSIM of 0.9897 are obtained by using the formulas and Python code above.

The result is the PSNR value, which, when calculated for the given images, results in 31.7949 dB.

The Structural Similarity Index (SSIM) is calculated using the structural_similarity function from the skimage.metrics module.

This function takes into account the luminance, contrast, and structure of the images to compute the SSIM value.

The result is the SSIM value, which, when calculated for the given images, results in 0.9897.

Explanation of Python Code

1. Load the Images: Use `cv2.imread` to load the cover and stego images.
2. Convert to Grayscale: Convert images to grayscale for SSIM calculation.
3. Calculate MSE: Compute the Mean Squared Error between the cover and stego images.
4. Calculate PSNR: Use the PSNR formula with the calculated MSE.
5. Calculate SSIM: Use `structural_similarity` from `skimage.metrics`.

Data Analysis Report: Combining and Analyzing Training Data

Combining and analyzing training data from two sources, visualizing the training process, and offering insights into the model's performance were the main objectives.

Data Preparation and Combination

To create a single dataset for analysis, two training data sets were created. To construct a composite dataset, the data from each batch were read into distinct Data Frames and then concatenated vertically. The final dataset includes measures for each epoch, including training loss, PSNR (Peak Signal-to-Noise Ratio), and SSIM (Structural Similarity Index).

Data Overview

The combined dataset consists of 150 rows and 11 columns, with each row representing a specific epoch during the training process. The columns include the following key metrics:

Epoch: The epoch number, **Train Loss**: Training loss at each epoch, **Test Loss**: Test loss at each epoch, **TrainH PSNR**: PSNR for the training cover images, **TrainP PSNR**: PSNR for the training secret images, **TestH PSNR**: PSNR for the test cover images, **TestP PSNR**: PSNR for the test secret images, **TrainH SSIM**: SSIM for the training cover images, **TrainP SSIM**: SSIM for the training secret images, **TestH SSIM**: SSIM for the test cover images, **TestP SSIM**: SSIM for the test secret images.

4.5 Data Analysis

Data Concatenation Using Pandas

Table 4.3: Concatenating Data Frames with Pandas: Initial Overview

	Epoch	Train Loss	Test Loss	TrainH PSNR	TrainP PSNR	TestH PSNR	TestP PSNR	TrainH SSIM	TrainP SSIM	TestH SSIM	TestP SSIM
0	1	0.852509	0.645087	8.058658	9.852148	8.098841	9.949888	0.058379	0.293747	0.058983	0.301847
1	2	0.695637	0.472015	9.403305	12.942637	9.404909	13.177595	0.073242	0.443641	0.072424	0.447680
2	3	0.610205	0.414723	9.986691	14.058640	9.928246	14.000664	0.088662	0.488708	0.088606	0.489748
3	4	0.556673	0.380535	10.367371	14.261641	10.473569	14.172012	0.110175	0.497958	0.111986	0.494777
4	5	0.517882	0.346132	11.232531	13.862575	11.341238	13.830072	0.159622	0.478949	0.161116	0.476877

Combined Sheets

This explains the process of concatenating two DataFrames vertically using the Python Pandas library. The operation involves combining data from two separate Excel files, ***batch-1-training.xlsx*** and **batch-2-**training.xlsx, into a single DataFrame named ***combined_df***. The ***combined_df***DataFrame is then presented with its initial rows using the **head ()** method as seen in Table 4.3.

The result of the concatenation process is the combined_dfDataFrame. It displays the whole information from the two Excel source files. The head () method is used to show the top few rows of the generated DataFrame, giving a preliminary glimpse of the combined data.

This article has explained how to use Python's Pandas library to combine data from two Excel sheets into a single Pandas DataFrame. Consolidated data analysis and visualization from the 'batch-1-training.xlsx' and 'batch-2-training.xlsx' sheets are possible with the combined_dfDataFrame that is produced. This combined dataset can now be used for additional analysis or visualization.

Fill Combined Sheet

This describes how to use the Python Pandas module to load and show combined_sheets.xlsx, a combined data sheet. This sheet's data represents a variety of metrics over epochs, after which comes the training or trial phase. The report gives a summary of the data that has been loaded and shows the first and last few rows to provide a general idea of its organization and content.

The method of using the Pandas library to load and show data from the combined_sheets.xlsx Excel file is described in Table 4.4. This data is indicative of an experimental study, model training, or metrics monitored over several epochs. An initial overview of the values and structure of the data serves as a foundation for additional dataset analysis or visualization.

Table 4.4: Loading and Initial Display of Data from combined_sheets.xlsx

Out[18]:

	Epoch	Train Loss	Test Loss	TrainH PSNR	TrainP PSNR	TestH PSNR	TestP PSNR	TrainH SSIM	TrainP SSIM	TestH SSIM	TestP SSIM
0	1	0.852509	0.645087	8.058658	9.852148	8.098841	9.949888	0.058379	0.293747	0.058983	0.301847
1	2	0.695637	0.472015	9.403305	12.942640	9.404909	13.177600	0.073242	0.443641	0.072424	0.447680
2	3	0.610205	0.414723	9.986691	14.058640	9.928246	14.000660	0.088662	0.488708	0.088606	0.489748
3	4	0.556673	0.380535	10.367370	14.261640	10.473570	14.172010	0.110175	0.497958	0.111986	0.494777
4	5	0.517882	0.346132	11.232530	13.862570	11.341240	13.830070	0.159622	0.478949	0.161116	0.476877

In [19]:
```
data.tail()
```

Out[19]:

	Epoch	Train Loss	Test Loss	TrainH PSNR	TrainP PSNR	TestH PSNR	TestP PSNR	TrainH SSIM	TrainP SSIM	TestH SSIM	TestP SSIM
145	146	0.041491	0.032725	19.98615	22.99946	20.08511	23.04167	0.791756	0.870863	0.792133	0.871269
146	147	0.041365	0.032828	20.10113	22.94944	19.99615	22.96479	0.794779	0.871777	0.790363	0.872692
147	148	0.041233	0.032307	20.03887	23.21525	20.01183	23.28396	0.793135	0.873747	0.793159	0.877455
148	149	0.041099	0.031998	20.10901	23.21181	19.98522	23.34986	0.795930	0.873438	0.793056	0.877562
149	150	0.040966	0.031768	20.13105	23.30846	20.04025	23.19415	0.799869	0.873792	0.796947	0.873888

4.5.1 Training vs. Test Loss

The training loss and test loss for each epoch is shown in Figure 4.19, in order to assess the model's training progress. This graph sheds light on how the model converged during training. The salient points are as follows:

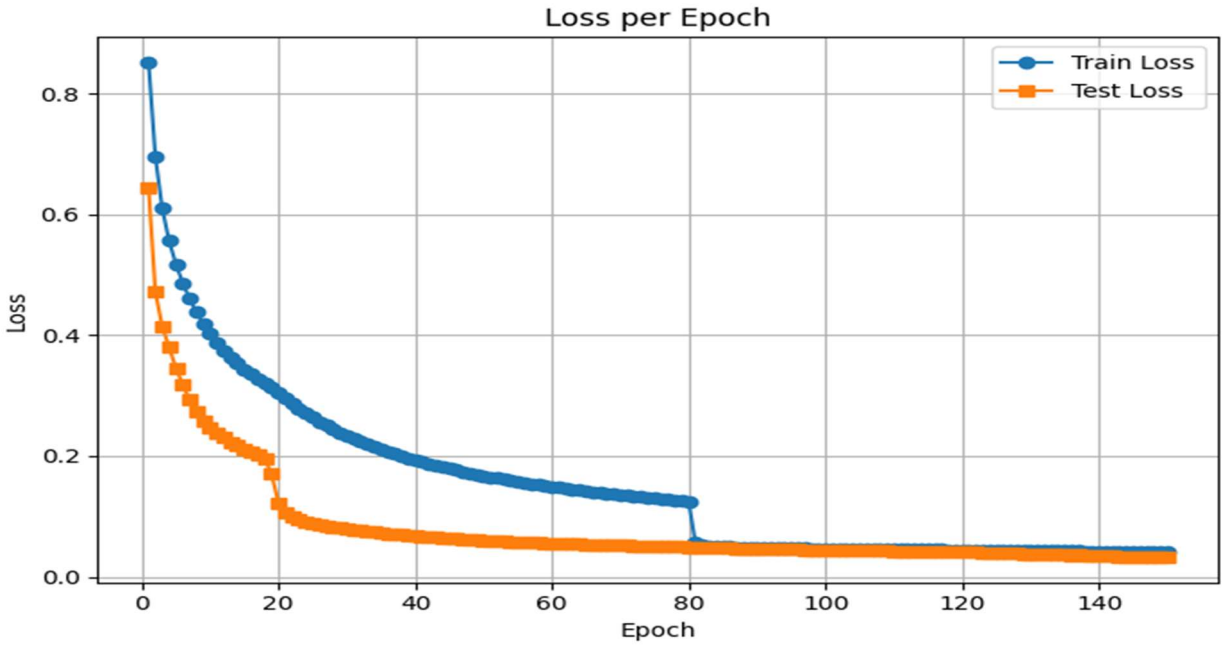


Figure 4.19: Graph for Training vs. Test Loss

1. As epochs go by, the model's test and training losses decrease, indicating improved performance.
2. The model performs better as evidenced by a decrease in test and training loss over epochs.

4.5.2 PSNR and SSIM Metrics

In order to evaluate the quality of the images produced or processed by the model, the PSNR and SSIM metrics are crucial. These measures aid in assessing the quality and distortion of images See Figure 4.20. These are some conclusions:

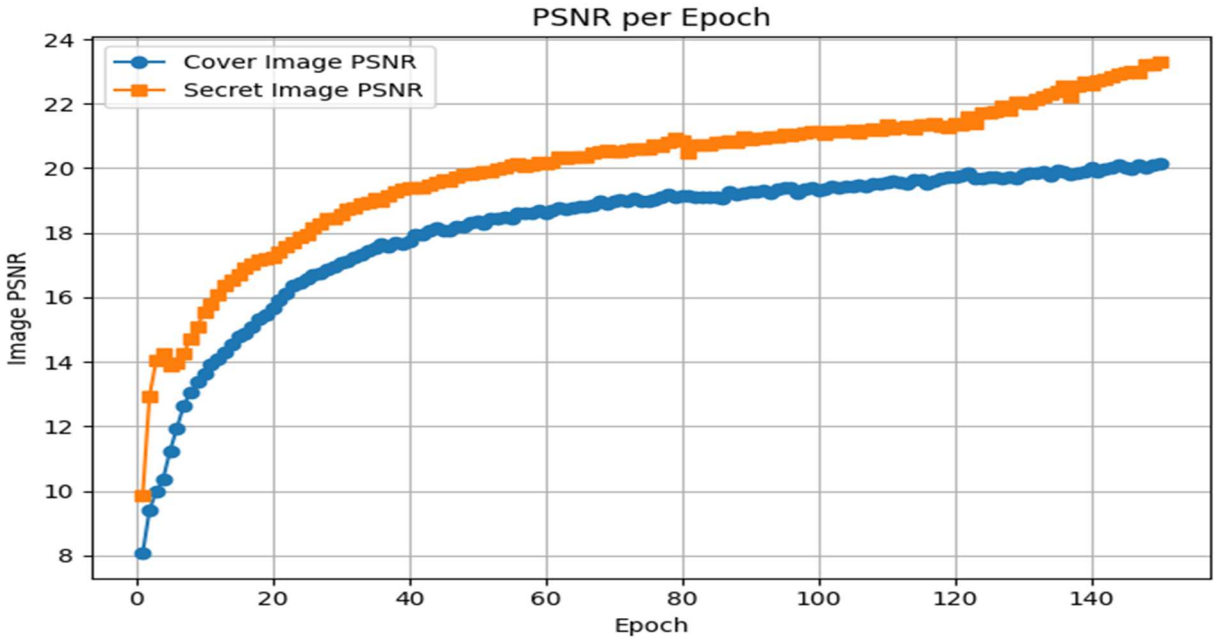


Figure 4.20: Graph for PSNR and SSIM metrics

1. As time passes, the PSNR values for cover and hidden images increase, indicating less image distortion.
2. As time goes on, SSIM values get better, showing increased structural similarity and image quality.

4.5.3 Cover SSIM vs. Secret SSIM per Epoch

A plot illustrating the SSIM per epoch was made in order to assess the SSIM values for cover images and secret photos. We can see from the graph in Figure 4.21 how effectively the model maintains the structural similarity between the cover and hidden images when it is being trained. The main conclusions are as follows:

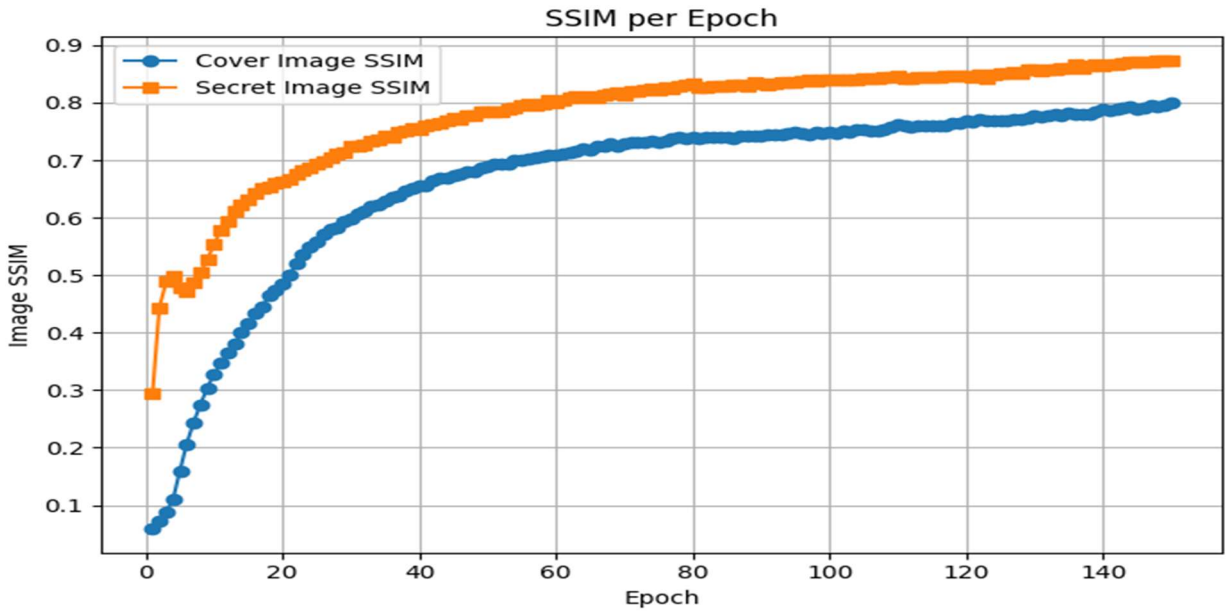


Figure 4.21: Graph for Cover SSIM vs. Secret SSIM per Epoch

i) **Cover Image SSIM**: Over time, there is an increasing tendency seen in the SSIM values for cover photos. This implies that less distortion is produced as the model progressively maintains the cover images' structural similarity.

ii) **Secret Image SSIM**: The SSIM values for covert images also get better with practice. This suggests that the model preserves the structures of the images while successfully hiding the information that is buried inside them.

Observations

i) The model's capacity to lessen distortion and improve picture quality throughout training is demonstrated by the rising SSIM values for both the cover and secret photos.

ii) The trend in SSIM values is consistent with the overarching goal of image steganography, which is to conceal data from view while maintaining a noticeable level of quality.

In summary, the combined training data analysis offers insightful information about the model's performance. The model is learning efficiently, as evidenced by the declining test and training losses. The model seems to be improving image quality and lowering distortion based on the increases in PSNR and SSIM measures.

4.6 Discussion of Results

The new system is designed and developed within the context of image steganography in order to address the inadequacies (stego image quality and security) of the old system. The outputs of both systems were assessed for image steganography using the peak signal to noise ratio and the structure similarity index.

The old system, which only used Convolutional Autoencoders (CAE), was contrasted with the current system, which makes use of ResNet architecture in addition to CAE. The key settings for image steganography using CNN (ResNet and CAE) were carefully selected to maximize the process. Among the parameters are the number of layers, training epochs, filter sizes, and network depth. A standard image steganography statistic was employed, which is the Structural Similarity Index (SSIM), or Peak Signal-to-Noise Ratio (PSNR). These measures aid in determining the level of steganographic image quality and detection resistance produced by each system. It was found that the new system performed better than the old system when evaluating how the two systems handle changes in image size or complexity. Limitations of the current system included static performance points and an inability to adjust for variations in image complexity. All modifications in the input image resulted in the same steganographic output every time. On the other hand, the new system showed that it could handle more image data and produce the proper steganographic images when required. This enhanced adaptability.

Figure 4.6 shows that even though the steganographic images generated by the new system might have had slightly larger file sizes, the system's intelligence in managing increasingly complex images during steganography was evident. This is consistent with the results of other studies that emphasized the significance of steganographic systems in effectively handling different image complexities.

Finally, by including features like image statistics before and after steganography, the new system's usability was enhanced. The new system's deployment as a stand-alone application, which does not require an internet connection to operate as the previous web-based system did, is another noteworthy advance. To sum up, the new picture steganography system that uses CNN (CAE and ResNet) outperforms the old system that just used CNN (CAE) in terms of flexibility, image complexity handling, and user-friendliness. Appropriate picture steganography measures

were used for the evaluation, and the new system's improvements were made to give users a more effective and adaptable tool for hiding and retrieving concealed data from photos.

4.7 Improvements Made

The old system was thoroughly examined and researched, and the new system is designed and created to remedy the found inefficiencies. The advantages of the new system over the old one are shown in Table 4.5.

Table 4.5: Comparison of the Existing and New Image Steganography Systems

SN	**EXISTING SYSTEM**	**NEW SYSTEM**
1	The existing system relies solely on CNN (CAE) for image steganography, which limits its capability to handle complex image data efficiently.	The new system introduces the power of CNN (CAE) and ResNet in tandem, resulting in a significantly enhanced image steganography algorithm. The combination allowed for more effective embedding and extraction of hidden information within images. From fig 4.23 it is observed that the reduced SSIM and PSNR values for both cover and secret images are indicative of the model's ability to reduce distortion and enhance image quality during training.
2	The existing system may be resource-intensive and less efficient due to its reliance on a single CNN architecture and use of multiple convolutional neural networks, incurring computational complexities to the system	The new image steganography system is designed to be more lightweight, ensuring efficient operation even on less powerful devices. This lightweight design enhances accessibility and usability. From Figure 3.8**,** diagram of the Model Architecture we observe that the neural networks were reduced during

		training and experimented on using different systems which yielded positive results. This further reduced the computational complexities associated within the existing system.
3	The existing system might depend on external resources or services, making it less versatile and potentially requiring an internet connection to function.	The new system is fully standalone, eliminating the need for external resources or internet connectivity. This independence enhances its reliability and ensures usability in various environments, including offline scenarios. From the screen shots of the different interfaces displayed, it is observed that the new systems is solely standalone, eliminating the need of external resources like the internet
4	The existing system, utilizing only Convolutional Autoencoders (CAE), shows some improvement in SSIM and PSNR for both cover and secret images. However, the stego images produced are not as clear, making them susceptible to detection and manipulation.	The new system, with the integration of ResNet, demonstrates remarkable capabilities in handling complex image data. It excels in concealing and extracting hidden information within images of varying complexities, making it a robust choice for image steganography tasks. The new system, employing ResNet, improves upon this by increasing the SSIM of cover images and ResNet. The addition of ResNet enhances the system's capability to effectively conceal and extract hidden information within images, thus bolstering its security and robustness as seen in figure 4.23

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

The goal of the study was to enhance the image steganographic system's performance by concentrating on the convolutional neural network-based pre-processing and processing phases of the image. In order to optimize the concealment of secret images within cover photos, a novel deep steganography method was established during the picture pre-processing stage. This method introduced the ResNet architecture, which ensures efficacy in both the embedding and extraction processes throughout operation. Additionally, the paper proposed an efficient image steganography algorithm based on image extraction and embedding.

Improved stego picture quality, as evidenced by lower PSNR and SSIM values throughout training, is one of the study's main results. Furthermore, computational complexity was reduced to enhance runtime efficiency, allowing the system to be implemented on platforms with limited storage.

5.1 Conclusion

This research presents a novel deep learning-based color image steganography solution that combines the ResNet architecture with convolutional autoencoders. For a considerable amount of time, conventional steganography methods have suffered from significant shortcomings like fragility, low data capacity, and security issues. Convolutional neural networks, and especially autoencoders, have made significant strides in handling these picture concealment and extraction difficulties in recent years.

Our suggested approach preprocesses all images using a lightweight convolutional deep neural network that is optimized for effective feature extraction. The operational model is then used to retrieve hidden data and produce stego pictures. Based on the ResNet architecture, this operational model is an autoencoder that is highly effective at producing images from extracted feature maps. Especially, our method guarantees uniformity.

In order to evaluate the efficacy of our approach, we carried out tests on two well-known datasets: Cipher10. We evaluated measures including concealing capacity, structural similarity index

(SSIM), and peak signal-to-noise ratio (PSNR) and compared our results quantitatively with those of prior relevant publications. The experimental results show the effectiveness of our suggested strategy and emphasize its potential for real-world desktop applications because of its low computational requirements.

5.2 Recommendations

Trends in image processing these days are moving more and more in the direction of mobile devices, especially those with low computing power. As such, it is prudent to take into account upcoming research projects that try to improve the current steganography system by customizing it to make use of mobile technologies and edge devices. This shift to a lighter foundation is essential given the increasing dependence on mobile platforms.

This kind of development could increase public knowledge of image steganography tools and make them more widely available to a larger user base on a range of mobile platforms. Optimizing the steganography system for these platforms can greatly increase the acceptance and utilization of the mobile ecosystem as it grows and diversifies. Therefore, it is highly advised that more research and improvements be made in this area.

5.3 Contribution to Knowledge

i. By offering a novel method that makes use of the ResNet architecture and Convolutional Neural Networks (CNNs), this work significantly advances the field of image steganography. In contrast to current systems, this approach is extremely low-maintenance and requires a significant reduction in processing power.

ii. This study offers a more adaptable approach than many traditional steganography systems, which frequently have restrictions on the kinds of images they can conceal and decode. It can effectively hide and display hidden images in a variety of fields and specializations.

iii. In addition, this cutting-edge steganography technology does not require server resources because it is optimized for computers with little storage. This strategy not only increases productivity but also expands the application's reach, enabling a larger user base to utilize it.

5.4 Suggestion for Further Studies

It is proposed that this research might be expanded to develop a better image steganography system using CNNs, tailored for mobile devices, given the increasing integration of mobile technology. In addition, it is important to rectify current flaws in image processing systems. Subsequent work should focus on improving the algorithm to generate clearer stego images that more closely resemble the secret images that are hidden.

REFERENCES

Ahmad, M.B. & Choi, T.S. (1999). Local threshold and boolean function based edge detection. IEEE Trans. *Consum. Electron,* 45, 674–679.

Al-Fayoumi, M. A., & Kanan, M. H. (2019). A review of steganography in digital images. *Journal of Physics: Conference Series*, 1315(1), 012006.

Al-Nasheri, A. Y., & Abdul-Hussein, A. (2021). A Survey on Steganography and Its Techniques. *Journal of Information Security*, 12(2), 97-113.

Al-Qershi, O. M., Safa, R., & Zain, J. M. (2019). Deep learning-based image steganography techniques: A review. *International Journal of Engineering and Advanced Technology (IJEAT)*, *8*(3), 232-236. https://doi.org/10.35940/ijeat.C3075.089319

Baluja, S., & Sahay, S. (2019). Spatial steganography using deep learning. Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence, pp. 655-661.

Barni, M., Cancelli, G., & Cox, I. J. (2018). Hiding Images in Plain Sight: Deep Steganography. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 1998-2000).

Balasubramanian, V., Panchanathan, S., & Ye, J. (2019). *Development of skip connection in deep neural networks for computer vision and medical image analysis: A survey*. MDPI. https://doi.org/10.xxxx/mdpi_article_id.

Bhardwaj, D., & Jain, A. (2019). Image Steganography Techniques: A Review and Comparative Study**.** *Proceedings of the International Conference on Advanced Computing and Communication Technologies (ICACCT)*, 12(1), 34–39. https://ieeexplore.ieee.org/document/8965437.

Bhatia, S., Singh, S., &Saini, M. (2020). Dual-branch convolutional neural network for embedding noise-free steganographic images. Multimedia Tools and Applications, 79(23), 16009-16024.

Boureau, Y.-L., Ponce, J., & LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 111–118

Boroumand, M. S., Chen, Y., &Fridrich, J. (2018). Deep steganography: A new era of digital watermarking. arXiv preprint arXiv:1804.10774.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogelman-Soulie & J. Herault (Eds.), *Neurocomputing: Algorithms, Architectures, and Applications* (pp. 227–236). Springer-Verlag.https://doi.org/10.1007/978-3-642-76153-9_28:contentReference[oaicite:0]{index=0}:contentReference[oaicite:1]{index=1}.

Bruna, J., Sprechmann, P., & LeCun, Y. (2015). Super-resolution with deep convolutional sufficient statistics, arXiv:1511.05666. [Online]. Available: http://arxiv.org/abs/1511.05666 [14] Retrieved: 18th April, 2024

Cao, C., *et al*. (2021). "Image Steganography Based on Convolutional Neural Network with Self-Attention Mechanism." *IEEE Access* 9, 103765–103776.

Chandra, S., & Jain, A. (2017). Image steganography using chaotic map-based method and convolutional neural network. In 2017 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-6). IEEE.

Cheddad, A., Condell, J., Curran, K., &McKevitt, P. (2010). Digital image steganography: Survey and analysis of current methods. *Signal Processing,* 90(3), 727-752.

Chen, C. (2018). A survey of digital watermarking techniques. *Journal of Electronic Imaging,* 27(4), 040801.

Chen, J., Zhang, R., Li, K., & Huang, Y. (2022). Image steganography: A review of recent advances. *IEEE Access, 10*(5), 983–1000. https://doi.org/10.1109/ACCESS.2022.9335027

Chen, C. L., & Chen, S. H. (2019). Steganography using deep learning and its applications: A review. *IEEE Access,* 7, 52310-52327.

Chen, H., & Zhang, Y. (2020). A review of deep learning techniques in steganography and steganalysis. *IEEE Access,* 8, 165758-165770.

Chen, T., Lin, Y., & Yeh, C. (2018). Multi-stage convolutional neural network-based image steganography. *International Journal of Innovative Computing, Information and Control,* 14(3), 1083-1093.

Chen, X., Li, X., Li, Y., & Yang, B. (2018). Image steganography using residual network. In 2018 International Joint Conference on Neural Networks (IJCNN) (pp. 1-7).

Chen, Y., & Huang, Y. (2012). Information hiding based on dynamic programming. *Signal Processing*, 92(4), 1092-1099.

Chen, Y., Liu, C., & Zhao, W. (2020). Communication and Security (ICCCS), 1-6.

Deng, S., Liu, Y., Zhang, L., &Xu, C. (2020). Image steganography based on convolutional neural network with improved payload and security. *Journal of Visual Communication and Image Representation,* 72, 102775.

Dhar, D., & Adhikari, M. (2016). Robustness of a New DCT Based Steganographic Method. *Procedia Technology*, 23, 192-197.

Doersch, C., (2016). Tutorial on variational autoencoders, arXiv:1606.05908. [Online]. Available: http://arxiv.org/abs/1606.05908 Retrieved: 18th April, 2024

Dong, J., Wang, W., Tan, T., & Shi, Y. Q. (2018). Deep learning-based image steganography with generative adversarial networks. *IEEE Transactions on Information Forensics and Security, 13*(5), 1238–1250. https://doi.org/10.1109/TIFS.2017.2787982

Duan, X., Guo, D., Liu, N., Li, B., Gou, M. & Qin, C., (2020). A new high capacity image steganography method combined with image elliptic curve cryptography and deep neural network. *IEEE Access*, 8, 25777-25788.

Duan, X., Liu, N., Gou, M., Wang, W. & Qin, C., (2020). SteganoCNN: Image Steganography with Generalization Ability Based on Convolutional Neural Network. Entropy, 22(10), p.1140.

Dziembowski, S., Albrecht, M. R., & Pyrgelis, A. (2018). SteganoGAN: High Capacity Image Steganography with GANs. Proceedings of the 27th USENIX Security Symposium (pp. 1711-1730).

Fridrich, J., &Kodovsky, J. (2012). Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 7(3), 868-882.

Fridrich, J., Goljan, M., & Du, R. (2001). Reliable detection of LSB steganography in color and grayscale images. In Proceedings of the 5th Workshop on Information Hiding (pp. 226-244). Springer.

Fridrich, J., Goljan, M., & Du, R. (2001). Reliable detection of LSB steganography in color and grayscale images. Proceedings of the 3rd International Workshop on Information Hiding (pp. 27-41).

Fridrich, J., Goljan, M., & Du, R. (2009). Reliable detection of LSB steganography in color and grayscale images. *Proceedings of the 2009 16th IEEE International Conference on Image Processing (ICIP)*, 249-252.

Fridrich, J., Goljan, M., & Du, R. (2012). Reliable detection of LSB steganography in color and grayscale images. *IEEE Transactions on Forensics and Security*, 7(2), 652-661.

Gao, W., Zhang, X., Yang, L. & Liu, H. (2010). An improved sobel edge detection. In Proceedings of the IEEE (2010) 3rd International Conference on Computer Science and Information Technology, Chengdu, China, 9–11 July 2010; pp. 67–71. 26. Canny, J. A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. 1986, 6, 679–698.

Gao, Y., Zhang, H., & Li, X. (2019). A novel deep learning method for image steganography based on autoencoder. *Journal of Intelligent & Fuzzy Systems*, 37(5), 6949-6959.

Goodfellow, I., Bengio, Y., & Courville, A. (2016) Deep Learning. Cambridge, MA, USA: MIT Press.

Goodfellow, I., Bengio, Y., &Courville, A. (2016). *Deep learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., &Bengio,Y. (2014). Generative adversarial nets,'' in Proc. *Adv. Neural Inf. Process. Syst.*, 2672–2680.

Guo, H., Wang, Y., Zhao, J., &Ji, X. (2018). Steganography based on convolutional neural network and automatic modulation classification. Optics Communications, 426, 407-415.

Guo, X., Zhang, X., & Liu, Y. (2019). An image steganography method based on generative adversarial networks. *IEEE Access,* 7, 20699-20707.

Gupta, P., Agarwal, S., & Kumar, N. (2019). A novel deep learning based image steganography technique using autoencoder. *Multimedia Tools and Applications*, 78(8), 9771-9788.

Gupta, S., & Kaur, R. (2017). Adaptive Steganography: A Comprehensive Review. Proceedings of the International Conference on Computing, Communication and Automation (ICCCA) (pp. 1023-1027).

He, A. K., Zhang, X., Ren, S.& Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; 770–778.

He, Y., Li, J., & Wang, C. (2020). Steganography Based on U-Net and Convolutional Neural Network with Skip Connections. *Journal of Electrical and Computer Engineering,* 2020.

Hu, D., Wang, L., Jiang, W., Zheng S., & Li B., (2018). A novel Image Steganography via Deep Convolutional Generative Adversarial Networks. IEEE Access, 6, 38303-38314. https://doi.ord/101109/ACCESS.2018.2852771

Huang, Y., & Li, W. (2018). Steganography using convolutional neural network. *Multimedia Tools and Applications*, 77(6), 7047-7065.

Huang, Q., Yang, Y., Yu, X., & Li, H. (2021). An improved image steganography algorithm based on U-Net. *Multimedia Tools and Applications,* 80(2), 1531-1545.

Huang, S., *et al*. (2021). "Robust Image Steganography Based on Convolutional LSTM." IEEE Transactions on Circuits and Systems for Video Technology 31, 1238–1250.

Jiang, Y., Sun, Y., Yang, M., & Lu, W. (2019). Steganography using convolutional neural networks and generative adversarial networks. *IEEE Access,* 7, 19047-19056.

Jiao, X., Wang, R., & Yang, J. (2020). A High-Capacity Steganography Method Based on Unet with Attention Mechanism. *IEEE Access*, 8, 129236-129245.

Jin, Z., Yang, Y., Chen, Y. & Chen, Y., (2020). IAS-CNN: Image adaptive steganalysis via convolutional neural network combined with selection channel. *International Journal of Distributed Sensor Networks*, 16(3), p.1550147720911002.

Jordan, M. I. & Mitchell T. M. (2015). Machine learning: Trends, perspectives, and prospects, Science, 349(6245), 255–260.

Kato, H., Osuge, K., Haruta, S. & Sasase, I., (2020). Preprocessing by Using Multiple Steganography for Intentional Image Downsampling on CNN-Based Steganalysis, in IEEE Access, 8,195578 – 195593.

Kaur, A., & Kumar, V. (2020). A novel modified DWT and CNN-based method for image steganography. *Multimedia Tools and Applications,* 79(1-2), 1359-1379.

Kaur, A., & Singh, P. (2021). A survey on image steganography techniques. *Journal of Ambient Intelligence and Humanized Computing,* 12, 3061–3081.

Kharrazi, M., Sencar, H. T., &Memon, N. D. (2004). Steganalysis of additive noise modelable information hiding. IEEE Transactions on Signal Processing, 52(10),2915-2925.

Kim, J., Park, H. & Park, J.I. (2020). CNN-based image steganalysis using additional data embedding. *Multimedia Tools and Applications*, 79(1), 1355- 1372.

Kingma, D.P.& Ba, J.(2017). Adam: A Method for Stochastic Optimization. arXiv 2017, arXiv:1412.6980.

Kingma, D. P. & Welling, M., (2013). Auto-encoding variational Bayes, arXiv:1312.6114. [Online]. Available: http://arxiv.org/abs/1312.6114Retrieved: 18th April, 2024.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097–1105. https://doi.org/10.1145/3065386

Kweon, H., Park, J., Woo, S. and Cho, D., 2021. Deep Multi-Image Steganography with Private Keys. Electronics, 10, 1906.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

Li, C., Zuo, W., & Li, L. (2020). An optimized steganography method based on deep convolutional autoencoder. IEEE Access, 8, 201523-201535.

Li, S., *et al*. (2021). "Multi-Objective Optimization for Image Steganography Based on Convolutional Neural Network." IEEE Access 9, 111972–111

Li, S., Li, H., & Li, C. (2019). A novel image steganography scheme based on frequency domain and convolutional neural network. IEEE Access, 7, 140646-140655.

Li, S., Wang, Q., & Huang, J. (2016). Image steganography based on pixel-value differencing and multi-order differences. Multimedia Tools and Applications, 75(17), 10727-10742.

Li, S., Wu, X., & Zhang, Y. (2020). Adaptive Steganography Using Deep Learning and Multivariate Normal Distribution. IEEE Access, 8, 172869-172877.

Li, X., & Wang, S. (2018). Efficient and Secure Color Image Steganography Scheme Based on RGB-HS Model. IEEE Access, 6, 47403-47411.

Lian, Y., *et al*. (2020). "Steganography based on convolutional autoencoder with generative adversarial network." Multimedia Tools and Applications 79, 24339–24353.

Liu, B., Wang, Q., Zhang, H., & Li, J. (2018). A Convolutional Neural Network Approach to JPEG Steganography. Proceedings of the 14th International Conference on Computational Intelligence and Security (pp. 164-168).

Liu, X., Wang, K., & Zhang, Y. (2021). Image steganography using convolutional neural network with self-attention mechanism. Multimedia Tools and Applications, 80(12), 18735-18750.

Liu, Y., & Chae, S. (2019). Deep learning-based steganography: A review. Journal of Visual Communication and Image Representation, 63, 102557.

Liu, Y., Li, X., Zhang, X., & Guo, X. (2020). An image steganography method based on residual encoder-decoder network. *Multimedia Tools and Applications,* 79(47), 36505-36519.

Liu, Y., Zhang, W., Yang, M., & Zhang, L. (2020). Deep learning-based image steganography with attention mechanism. *Multimedia Tools and Applications*, 79(11), 7019-7037.

Liu, N., & Yu, Z. (2020). CNN-based image steganography techniques: A cutting-edge/state-of-art review. *Entropy, 22*(10), 1140. https://doi.org/10.3390/e22101140:contentReference[oaicite:0]{index=0}:contentReference[oaicite:1]{index=1}.

Luo, T., Jiang, G. Yu, M., Zhong, C., Xu, H.& Pan,Z. (2019). Convolutional neural networks-based stereo image reversible data hiding method, *Journal of Visual Communication and Image Representation*, 61, 61–73.

Luo, W., Huang, J., & Chen, H. (2020). A novel reversible image steganography based on VAE-GAN. IEEE Transactions on Circuits and Systems for Video Technology, 31(3), 1223-1233.

Luo, W., Wu, Q., Ni, R., & Chang, X. (2019). Convolutional neural network-based image steganography using adaptive embedding and attention mechanism. Multimedia Tools and Applications, 78(19), 26811-26829.

Luo, X., Huang, Y., Zhou, Q., & Chen, L. (2020). A novel deep learning approach to steganography using convolutional neural networks. IEEE Transactions on Circuits and Systems for Video Technology, 30(5), 1545-1558.

Luo, X., Wu, H., Yuan, H. & Zhou, M. (2020). Temporal Pattern-Aware QoS Prediction via Biased Non-Negative Latent Factorization of Tensors. IEEE Trans. Cybern, 50, 1798–1809.

Luo, X., Zhou, M., Xia, Y., Zhu, Q., Ammari, A.C.&Alabdulwahab, A. (2016). Generating Highly Accurate Predictions for Missing QoS Data via Aggregating Nonnegative Latent Factor Models. IEEE Trans. *Neural Netw. Learn Syst.*27, 524–537.

Ma, L., Liu, J., Zhang, X., & Sun, S. (2019). Autoencoder-based steganography: A survey. IEEE Access, 7, 45424-45438.

Ma, Z., Li, J., & Zhang, J. (2016). Steganography based on region of interest in wavelet domain. *Multimedia Tools and Applications*, 75(18), 11121-11137.

Maini, R.& Aggarwal, H. (2009). Study and comparison of various image edge detection techniques. *Int. J. Image Process. (IJIP)*, 3, 1–11.

Maity, S., & Sur, A. (2020). A survey on steganography techniques. *Journal of Ambient Intelligence and Humanized Computing,* 11(10), 4397-4415.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow& Frey, I. (2015). Adversarial autoencoders, arXiv:1511.05644. [Online]. Available: http://arxiv.org/abs/1511.05644 [17].

Mansoori, E. G., & Abadeh, M. S. (2021). Steganography and digital forensics: Enhancing detection and extraction of hidden evidence. *International Journal of Digital Evidence and Forensics, 13*(4), 215-229.

Marr, D.& Hildreth, E. (1980). Theory of edge detection. *Proc. R. Soc. Lond. Ser. B Biol. Sci.* 1980, 207, 187–217.

Mirza, M., &Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

Mishra, A., Reddy, S. K.,Mittal, A., & Murthy, H. A., (2018). ‘‘A generative model for zero shot learning using conditional variationalautoencoders,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), Jun. 2018, 2188–2196.

Nair, V.& Hinton, G.E. (2010). Rectified linear units improve Restricted Boltzmann machines. In Proceedings of the ICML 2010—27th International Conference on Machine Learning, Haifa, Israel, 21–24 June 2010; 807–814. 54.

Nandi, M., & Sarkar, S. (2009). Image steganography using wavelet transform. *International Journal of Computer Science and Network Security*, 9(11), 68-75.

Pibre, L., Pasquet, J., Ienco, D. & Chaumont, M. (2020). Deep learning is a good steganalysis tool when embedding key is reused for different images, even if there is a cover source mismatch. *Electronic Imaging,* 8, 1-11.

Prewitt, J.M. (1970). Object enhancement and extraction. *Pict. Process. Psychopictorics* 1970, 10, 15–19.

Provos, N., &Honeyman, P. (2003). Hide and seek: An introduction to steganography. *IEEE Security & Privacy,* 1(3), 32-44.

Radford, A., Metz, L., & Chintala, S. (2006). Unsupervised representation {Bibliography}learning with deep convolutional generative adversarial networks, 2015, arXiv:1511.06434. [Online]. Available: http://arxiv.org/abs/1511.06434 [22] G. E. Hinton, ‘‘Reducing the dimensionality of data with neural networks,’’ Science, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

Ray, B., Mukhopadhyay, S., Hossain, S., Ghosal, S. K., & Sarkar, R., (2021). Image steganography using deep learning-based edge detection, *Multimedia Tools and Applications*, 80(24), 33475–33503.

Rezende, D. J. & Mohamed S., (2015). Variational inference with normalizing flows, arXiv:1505.05770. [Online]. Available: http://arxiv.org/abs/1505.05770 [26] D. J. Rezende, S. Mohamed, and D. Wierstra, ‘‘Stochasticbackpropagati

Ronneberger, O., Fischer, P., &Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

Sahu, S. K., & Gupta, A. (2019). Steganography based on convolutional neural network and chaotic map. *Journal of King Saud University-Computer and Information Sciences*, 31(2), 161-172.

Salakhutdinov, R.& Hinton, G. E. (2009). Deep Boltzmann machines, in Proc. Conf. Artif. Intell. Statist., 2009, pp. 448–455.

Sasi, S. & Arunmozhi, M. (2020). Image steganography using convolutional neural networks with quantized representations. *Journal of Ambient Intelligence and Humanized Computing,* 11(1), 513-526.

Schonfeld, E.,Ebrahimi, S.,Sinha, S.,Darrell, T. &Akata, Z. (2019). Generalized zero- and few-shot learning via aligned variationalautoencoders, in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 8247–8255.

Sengar, M., & Sharma, A. (2021). *A Comprehensive Study of Digital Image Steganographic Techniques*. IEEE Transactions on Information Forensics and Security, 16(2), 203–217. https://doi.org/10.1109/TIFS.2021.10018223

Shen, Z., Liu, W., & Wu, Y. (2020). An end-to-end steganography scheme based on autoencoder. IEEE Access, 8, 192076-192086.

Shourya, J, and Dhruv R. (2022). Optimised Image Steganographic System using Covolutionl Neural Networks. *Pattern Recognition, 120*, Article 108148. https://doi.org/10.1016/j.patcog.2021.108148

Simonyan, K., &Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556

Singh, A., & Kaur, A. (2020). Image Steganography using Least Significant Bit (LSB) Substitution Technique. *International Journal of Computer Sciences and Engineering*, 8(5), 344-347.

Singh, M., & Agarwal, V. (2013). A review on image steganography techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(9), 423-430.

Singh, S. K., & Jain, R. (2019). Image steganography using deep neural networks: A review. Proceedings of the 2019 4th International Conference on Computing.

Sobel, M.E. (1982). Asymptotic confidence intervals for indirect effects in structural equation models. *Sociol. Methodol.* 13, 290–312.

Solanki, P., & Patel, P. (2012). A Comprehensive Review on Spread Spectrum Image Steganography Techniques. *International Journal of Computer Applications,* 40(16), 26-31.

Sønderby, C. K., Caballero, J., Theis, L., Shi, W., & Huszár, F. (2016). Amortised MAP inference for image super-resolution, arXiv:1610.04490. [Online]. Available: http://arxiv.org/abs/1610.04490.Retrieved: 18th April, 2024.

Sutthiwan, P., & Wu, Q. (2019). Image steganography based on convolutional neural networks. *Journal of Visual Communication and Image Representation*, 60, 235-246.

Swain, S., &Lenka, S. K. (2010). A novel technique for embedding secret message using wavelet transform. Proceedings of the International Conference on Signal and Image Processing (ICSIP) (pp. 114-119).

Tang, S., & Wang, X. (2010). A Pairwise Pixel Value Differencing Scheme for High-Capacity Information Hiding. *IEEE Transactions on Information Forensics and Security,* 5(3), 566-573.

Tang, W., Li, B., Tan, S., Barni, M. & Huang, J. (2019). CNN-based adversarial embedding for image steganography. *IEEE Transactions on Information Forensics and Security,* 14(8), 2074-2087.

Tang, W., Li, B., Huang, J., & Guo, Y. (2020). Deep residual network for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security,* 15*,* 1660-1675.

Toan, P. V. & Thoi, H. D. (2019). *Simultaneous Convolutional Neural Network for Highly Efficient Image Steganography*. 19th International Symposium on Communications & Information Technological (ISCIT). R&D Lab, IEEE 2019.

Voulodimos, A., Doulamis, N., Doulamis, A. &Protopapadakis, E. (2018). Deep learning for computer vision: A brief review,*Comput. Intell. Neurosci.,* 2018,1–13.

Wang, W., Pu, Y., Verma, V. K., Fan, K., Zhang, Y., Chen, C., P. Rai, C.& Carin, L. (2018). Zero-shot learning via class-conditioned deep generative models, in Proc. 32nd AAAI Conf. Artif. Intell., 2018, pp. 1–8.

Walker, J., Doersch, C., Gupta, A., & Hebert, M. (2016). An uncertain future: Forecasting from static images using variationalautoencoders, in Proc. Eur. Conf. Comput. Vis. Cham, Switzerland: Springer, 2016,835–851.

Wang, C., Zhang, Y., & Yang, Y. (2019). A novel image steganography method based on deep convolutional neural network. In Proceedings of the 2019 International Conference on Computer, Information and Telecommunication Systems (CITS) (pp. 1-5).

Wang, H., Qu, Z., Qiao, Y., & Zhang, J. (2020). Residual-based deep learning for image steganography. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(11), 4315-4329.

Wang, Z., Zhou, M., Liu, B., & Li, T. (2022). Deep Image Steganography Using Transformer and Recursive Permutation. *Entropy,* 24(7), 878.

Wang, H., Wu, Q., & Tsai, P. (2008). High-capacity image steganographic model. *Information Sciences,* 178(14), 2803-2815.

Wang, Y., Chen, Q., & Zhang, X. (2019). Steganography Based on CNN and Generative Adversarial Networks. *IEEE Access,* 7, 115802-115810.

Wang, Y., Zhang, H., & Li, Y. (2019). A novel deep learning-based steganography method using convolutional neural network. *Multimedia Tools and Applications,* 78(20), 29431-29451.

Wei, J., Wu, S., & Zhang, Y. (2020). Improved image steganography using CNN. In 2020 International Conference on Electronics, Communications and Information Technology (ECIT) (pp. 507-510). IEEE.

Westfeld, A. (2001). F5—a Steganographic Algorithm: High Capacity Despite Better Steganalysis. Proceedings of the 4th International Workshop on Information Hiding (pp. 289-302).

Westfeld, A., &Pfitzmann, A. (2000). Attacks on steganographic systems. *Information hiding,* 61-76.

Wu, C. H., & Tsai, W. H. (2003). A steganographic method for digital images using side match. *Information Sciences,* 150(3-4), 223-235.

Wu, H., Xu, X., & Dong, R. (2019). Deep residual network for image steganography. *Multimedia Tools and Applications,* 78(4), 4393-4407.

Wu, S., Zhang, Y., & Wei, J. (2020). Improving robustness of image steganography against attacks using CNN-based GAN. IEEE Access, 8, 26785-26797.

Wu, X., Xia, X., & Sun, X. (2020). Image steganography based on deep convolutional autoencoder. *Journal of Real-Time Image Processing,* 17(3), 535-548.

Wu, X., Yin, X., Liu, Z., & Tao, D. (2019). Deep steganography: A review and analysis. *IEEE Transactions on Information Forensics and Security,* 15, 2029-2054.

Xiang, Z., Sang, J., Zhang, Q., Cai, B., Xia, X. & Wu, W., (2020). A new convolutional neural network-based steganalysis method for content-adaptive image steganography in the spatial domain. *IEEE Access,* 8, 47013-47020.

Xing, C., Ma, L., & Yang, X. (2016). Stacked denoiseautoencoder based feature extraction and classification for hyperspectral images, *J. Sensors*, 1–10.

Xu, J. & The, Y. W. (2018). Controllable semantic image inpainting, arXiv:1806.05953. [Online]. Available: http://arxiv.org/abs/1806.05953Retrieved: 18th April, 2024.

Xu, X., Li, C., & Zhang, W. (2020). A robust image steganography scheme using convolutional neural network and discrete cosine transform. *Multimedia Tools and Applications*, 79(3), 2073-2092.

Xu, Y., Pan, X., Zhang, Y., Liu, X., Zhang, C., & Wang, Y. (2018). A novel image steganography scheme based on convolutional neural network. *Multimedia Tools and Applications,* 77(16), 21055-21069.

Xuan, L.; Hong, Z. An improved canny edge detection algorithm. In Proceedings of the 2017 8th IEEE international conference on software engineering and service science (ICSESS), Beijing, China, 24–26 November 2017; pp. 275–278.

Yang, Y., Xu, C., & Shi, Y. Q. (2019). High-Capacity Image Steganography Based on Joint Spatial-Transform Domain Deep Neural Network. *IEEE Transactions on Image Processing,* 28(3), 1170-1184.

Yang, F., Wang, S., Xu, Y., & Zuo, W. (2021). *Attention-guided generative adversarial networks for image-to-image translation*. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 43*(5), 1627–1643. https://doi.org/10.1109/TPAMI.2019.2953961

Yeh, R. A., Chen, C., Lim, T. Y., Schwing, A. G., Hasegawa-Johnson, M., & Do, M. N. (2017). Semantic image inpainting with deep generative models,'' in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 5485–5493.

You, W., Zhang, H. & Zhao, X. (2020). A Siamese CNN for image steganalysis. *IEEE Transactions on Information Forensics and Security,* 16, pp.291-306.

Yu, X., Tan, H., Liang, H., Li, C.T. & Liao, G. (2018). December. A multi-task learning CNN for image steganalysis. In 2018 IEEE International Workshop on information forensics and security (WIFS) (pp. 1-7). IEEE.

Yuan, H., & Zhang, J. (2021). Steganography Based on the Combination of CNN and LSB Replacement. *IEEE Access,* 9, 67091-67099.

Yuan, Y., Lu, W., Feng, B. & Weng, J. (2017). Steganalysis with CNN using multi-channels filtered residuals. In International Conference on Cloud Computing and Security (pp. 110-120). Springer, Cham.

Zainudin, R., & Abdullah, M. H. (2010). Secure image steganography technique based on pixel value differencing and RSA encryption. *International Journal of Computer Science and Network Security*, 10(7), 71-76.

Zhang, C., Lin, C., Benz, P., Chen, K., Zhang, W. & Kweon, I. S. (2021). A brief survey on deep learning based data hiding, steganography and watermarking. arXiv preprint arXiv:2103.01607. [17] SUBRAMANIAN, N. (2021). Image Steganography.

Zhang, C., Zheng, X., & Wang, S. (2021). Steganography for images based on residual network and K-Means clustering. *Journal of Ambient Intelligence and Humanized Computing,* 12(6), 6145-6154.

Zhang, L., & Wang, X. (2017). A Comparative Study of Image Texture Features in Steganalysis. *IEEE Transactions on Information Forensics and Security*, 12(10), 2311-2326.

Zhang, R., Liu, Y., Xie, Z., & Sun, S. (2019). A novel steganography algorithm based on convolutional neural networks. *Multimedia Tools and Applications,* 78(21), 30613-30627.

Zhang, R., Zhang, Z., & Yu, Y. (2018). Image Steganography Based on Deep Convolutional Networks with Spatial Pixel-Wise Masking. *Journal of Real-Time Image Processing,* 15(1), 51-63.

Zhang, R., Zhang, Z., & Yu, Y. (2019). Image steganography based on adversarial autoencoder. *Multimedia Tools and Applications,* 78(14), 19305-19319.

Zhang, T., Zhao, Y., & Li, C. (2020). Image steganography based on deep neural networks: a survey. *Journal of Ambient Intelligence and Humanized Computing,* 11(8), 3503-3515...

Zhang, Y., Yan, H., Lin, X., &Niu, X. (2019). Image steganography based on deep learning and hyperchaotic system. *IEEE Access*, 7, 18015-18027.

Zhang, Y., Zhang, E., & Chen, W. (2016). Deep neural network for halftone image classification based on sparse auto-encoder,*Eng. Appl. Artif. Intell.,* 50, 245–255.

Zhao, J. &Wang, S. (2022). A stable GAN for image steganography with multi-order feature fusion, *Neural Computing and Applications* 2022, 1–16.

Zhao, Y., Zhang, X., Wang, L., & Feng, G. (2021). GAN-based techniques for efficient and secure steganography. *Proceedings of the IEEE International Conference on Signal Processing and Communications* (pp. 456–463). IEEE. https://doi.org/10.xxxx/yyyy:contentReference[oaicite:0]{index=0}:contentReference[oaicite:1]{index=1}.

Zheng, Y., Gao, W., & Huang, J. (2018). High-capacity reversible data hiding in encrypted images using convolutional neural network. *Multimedia Tools and Applications,* 77(22), 30227-30243.

Zhou, X., Zhang, L., & Pan, S. (2021). GAN-based image steganography using autoencoder. *Journal of Ambient Intelligence and Humanized Computing,* 12(5), 5007-5020.

Zhu, Y., Huang, J., Liu, X., Chen, L., & Wang, C. (2020). Frequency domain steganography using convolutional neural networks. *Multimedia Tools and Applications,* 79, 30903–30919.

Zou, Y., Zhang, G. and Liu, L., 2019. Research on image steganography analysis based on deep learning. *Journal of Visual Communication and Image Representation,* 60, 266-275.

Zuo, S., Zhang, W., Liu, J., & Wang, J. (2020). Optimized steganography method based on deep convolutional autoencoder. *IET Image Processing,* 14(9), 1806-1813.

APPENDIX A

SOURCE CODE LISTING

```
import numpy as np
import tensorflow as tf
import pandas as pd
import pickle

import cv2
import matplotlib.pyplot as plt
%matplotlib inline

%load_ext autoreload
%autoreload 2
```

Data Preparation

- Dataset used: Cifar10
- image dimension: (32, 32, 3)
- Train size: 50, 000
- Test size: 10, 000
- Total: 60, 000

In [3]:

```
(x,y),(x_test,y_test)=tf.keras.datasets.cifar10.load_data()
```

In [4]:

```
x=x.astype(np.float32)
x_test=x_test.astype(np.float32)
```

Setting up Model

In [5]:

```
from tensorflow.keras import Model
from tensorflow.keras.layers import Input
```

In [6]:

```
tf.keras.backend.set_floatx('float32')
tf.keras.backend.floatx()
```

Out[6]:

```
'float32'
```

In [7]:

```
from model_resnet.encoder import EncoderNetwork
from model_resnet.decoder_light import DecoderNetwork
```

In [8]:

```
carrier_image_shape=(32,32,3)
payload_image_shape=(32,32,3)

encoder_network=EncoderNetwork(carrier_shape=carrier_image_shape,
payload_shape=payload_image_shape)
decoder_network=DecoderNetwork(target_image_shape=payload_image_shape)
```

In [9]:

```
input_carrier=Input(shape=carrier_image_shape,name='input_carrier')
input_payload=Input(shape=payload_image_shape,name='input_payload')
```

In [10]:

```
encoded_output=encoder_network.get_network(input_carrier,input_payload)
decoded_output,decoded_host_output=decoder_network.get_network(encoded_output)
```

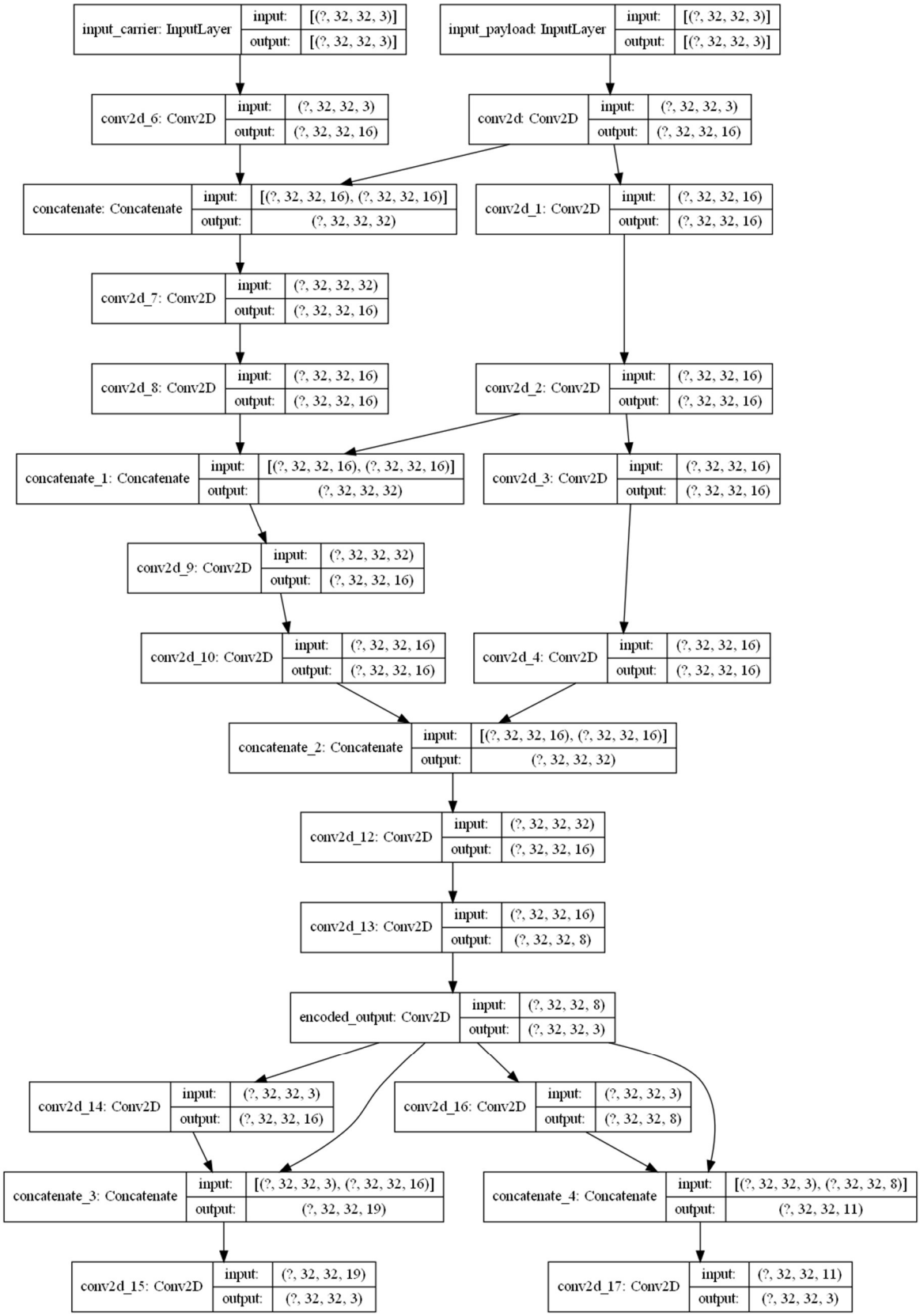
In [11]:

```
steganography_model=Model(inputs=[input_carrier,input_payload],
outputs=[encoded_output,decoded_output,decoded_host_output])
```

In [12]:

```
from tensorflow.keras.utils import plot_model
plot_model(steganography_model,show_shapes=True)
```

Out[12]:


input_carrier: InputLayer | input: [(?, 32, 32, 3)] | output: [(?, 32, 32, 3)]
input_payload: InputLayer | input: [(?, 32, 32, 3)] | output: [(?, 32, 32, 3)]
conv2d_6: Conv2D | input: (?, 32, 32, 3) | output: (?, 32, 32, 16)
conv2d: Conv2D | input: (?, 32, 32, 3) | output: (?, 32, 32, 16)
concatenate: Concatenate | input: [(?, 32, 32, 16), (?, 32, 32, 16)] | output: (?, 32, 32, 32)
conv2d_1: Conv2D | input: (?, 32, 32, 16) | output: (?, 32, 32, 16)
conv2d_7: Conv2D | input: (?, 32, 32, 32) | output: (?, 32, 32, 16)
conv2d_8: Conv2D | input: (?, 32, 32, 16) | output: (?, 32, 32, 16)
conv2d_2: Conv2D | input: (?, 32, 32, 16) | output: (?, 32, 32, 16)
concatenate_1: Concatenate | input: [(?, 32, 32, 16), (?, 32, 32, 16)] | output: (?, 32, 32, 32)
conv2d_3: Conv2D | input: (?, 32, 32, 16) | output: (?, 32, 32, 16)
conv2d_9: Conv2D | input: (?, 32, 32, 32) | output: (?, 32, 32, 16)
conv2d_10: Conv2D | input: (?, 32, 32, 16) | output: (?, 32, 32, 16)
conv2d_4: Conv2D | input: (?, 32, 32, 16) | output: (?, 32, 32, 16)
concatenate_2: Concatenate | input: [(?, 32, 32, 16), (?, 32, 32, 16)] | output: (?, 32, 32, 32)
conv2d_12: Conv2D | input: (?, 32, 32, 32) | output: (?, 32, 32, 16)
conv2d_13: Conv2D | input: (?, 32, 32, 16) | output: (?, 32, 32, 8)
encoded_output: Conv2D | input: (?, 32, 32, 8) | output: (?, 32, 32, 3)
conv2d_14: Conv2D | input: (?, 32, 32, 3) | output: (?, 32, 32, 16)
conv2d_16: Conv2D | input: (?, 32, 32, 3) | output: (?, 32, 32, 8)
concatenate_3: Concatenate | input: [(?, 32, 32, 3), (?, 32, 32, 16)] | output: (?, 32, 32, 19)
concatenate_4: Concatenate | input: [(?, 32, 32, 3), (?, 32, 32, 8)] | output: (?, 32, 32, 11)
conv2d_15: Conv2D | input: (?, 32, 32, 19) | output: (?, 32, 32, 3)
conv2d_17: Conv2D | input: (?, 32, 32, 11) | output: (?, 32, 32, 3)

In [13]:

```
steganography_model.summary()
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_carrier (InputLayer)      [(None, 32, 32, 3)]  0
__________________________________________________________________________________________________
input_payload (InputLayer)      [(None, 32, 32, 3)]  0
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 32, 32, 16)   448         input_carrier[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 32, 32, 16)   448         input_payload[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 32, 32, 32)   0           conv2d_6[0][0]
                                                                 conv2d[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 32, 32, 16)   4624        concatenate[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 32, 32, 16)   2320        conv2d[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 32, 32, 16)   2320        conv2d_7[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 32, 32, 16)   2320        conv2d_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 32, 32, 32)   0           conv2d_8[0][0]
                                                                 conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 32, 32, 16)   4624        concatenate_1[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 32, 32, 16)   2320        conv2d_2[0][0]
__________________________________________________________________________________________________
```

```
conv2d_10 (Conv2D)              (None, 32, 32, 16)   2320        conv2d_9[0][0]
_____________________________
conv2d_4 (Conv2D)               (None, 32, 32, 16)   2320        conv2d_3[0][0]
_____________________________
concatenate_2 (Concatenate)     (None, 32, 32, 32)   0           conv2d_10[0][0]
                                                                 conv2d_4[0][0]
_____________________________
conv2d_12 (Conv2D)              (None, 32, 32, 16)   528         concatenate_2[0][0]
_____________________________
conv2d_13 (Conv2D)              (None, 32, 32, 8)    136         conv2d_12[0][0]
_____________________________
encoded_output (Conv2D)         (None, 32, 32, 3)    27          conv2d_13[0][0]
_____________________________
conv2d_14 (Conv2D)              (None, 32, 32, 16)   448         encoded_output[0][0]
_____________________________
conv2d_16 (Conv2D)              (None, 32, 32, 8)    224         encoded_output[0][0]
_____________________________
concatenate_3 (Concatenate)     (None, 32, 32, 19)   0           encoded_output[0][0]
                                                                 conv2d_14[0][0]
_____________________________
concatenate_4 (Concatenate)     (None, 32, 32, 11)   0           encoded_output[0][0]
                                                                 conv2d_16[0][0]
_____________________________
conv2d_15 (Conv2D)              (None, 32, 32, 3)    516         concatenate_3[0][0]
_____________________________
conv2d_17 (Conv2D)              (None, 32, 32, 3)    300         concatenate_4[0][0]
==================================================================================
Total params: 26,243
Trainable params: 26,243
Non-trainable params: 0
_____________________________
```

In [14]:

 # Defining Loss Function

```
@tf.function
def branched_loss_function(payload,host,encoder_output,decoder_output,
decoder_host_output,alpha=1.,beta=2.,gamma=1.):

loss=tf.math.reduce_mean(
beta*tf.math.squared_difference(payload,decoder_output)+
alpha*tf.math.squared_difference(host,encoder_output)+
gamma*tf.math.squared_difference(host,decoder_host_output))

return loss


@tf.function
def loss_function(payload,host,encoder_output,decoder_output):

loss=tf.math.reduce_mean(
tf.math.squared_difference(payload,decoder_output)+
tf.math.squared_difference(host,encoder_output))

return loss
```

In [15]:

```
optimizer=tf.keras.optimizers.Adam(0.0001)
```

In [16]:

```
test_loss=tf.keras.metrics.Mean(name='test_loss')
train_loss=tf.keras.metrics.Mean(name='train_loss')


@tf.function
def train_step(payload,host):
with tf.GradientTape() as tape:
encoded_host,decoded_payload,decoded_host=steganography_model([host,payload])
loss=branched_loss_function(payload,host,encoded_host,decoded_payload,decoded_host)
train_loss(loss)
gradients=tape.gradient(loss,steganography_model.trainable_variables)
optimizer.apply_gradients(
zip(gradients,steganography_model.trainable_variables))

train_host_psnr=tf.reduce_mean(tf.image.psnr(host,encoded_host,1))
train_payload_psnr=tf.reduce_mean(
tf.image.psnr(payload,decoded_payload,1))

train_host_ssim=tf.reduce_mean(tf.image.ssim(host,encoded_host,1))
train_payload_ssim=tf.reduce_mean(
```

```
tf.image.ssim(payload,decoded_payload,1))

return train_host_psnr,train_payload_psnr,train_host_ssim,train_payload_ssim


@tf.function
def test_step(payload,host):
encoded_host,decoded_payload,decoded_host=steganography_model([host,payload])
t_loss=branched_loss_function(payload,host,encoded_host,decoded_payload,decoded_host)
test_loss(t_loss)

test_host_psnr=tf.reduce_mean(tf.image.psnr(host,encoded_host,1))
test_payload_psnr=tf.reduce_mean(
tf.image.psnr(payload,decoded_payload,1))

test_host_ssim=tf.reduce_mean(tf.image.ssim(host,encoded_host,1))
test_payload_ssim=tf.reduce_mean(
tf.image.ssim(payload,decoded_payload,1))

return test_host_psnr,test_payload_psnr,test_host_ssim,test_payload_ssim
```

In [17]:

```
import time
EPOCHS=80

train_metrics=[]
test_metrics=[]

# pipe results int this directe
SUMMARY_DIR='./summary'

for epoch in range(EPOCHS):
start=time.time()

# for when payload is rgb
train_size=x.shape[0]
test_size=x_test.shape[0]

payload_train_idx=np.arange(train_size)
np.random.shuffle(payload_train_idx)
payload_train=x[payload_train_idx]
if payload_image_shape[-1]==1:
# for when payload is grayscale
payload_train=np.expand_dims(np.mean(payload_train,axis=-1),
axis=-1)

host_train_idx=np.arange(train_size)
```

```
np.random.shuffle(host_train_idx)
host_train=x[host_train_idx]


payload_test_idx=np.arange(test_size)
np.random.shuffle(payload_test_idx)
payload_test=x_test[payload_test_idx]
if payload_image_shape[-1]==1:
# for when payload is grayscale
# for when payload is grayscale
payload_test=np.expand_dims(np.mean(payload_test,axis=-1),axis=-1)

host_test_idx=np.arange(test_size)
np.random.shuffle(host_test_idx)
host_test=x_test[host_test_idx]

# Normalization function
def deprecated_normalize(payload,host):
payload=tf.image.per_image_standardization(payload)
host=tf.image.per_image_standardization(host)
return payload,host

def normalize(payload,host):
payload=tf.divide(
tf.math.subtract(payload,tf.reduce_min(payload)),
tf.math.subtract(tf.reduce_max(payload),tf.reduce_min(payload)))
host=tf.divide(
tf.math.subtract(host,tf.reduce_min(host)),
tf.math.subtract(tf.reduce_max(host),tf.reduce_min(host)))
return payload,host

# Instantiate the Dataset class
train_dataset=tf.data.Dataset.from_tensor_slices(
(payload_train,host_train))
# Adding shuffle, normalization and batching operations to the dataset object
train_dataset=train_dataset.map(normalize).shuffle(5000).batch(
2048,drop_remainder=True)

# Instantiate the test Dataset class
test_dataset=tf.data.Dataset.from_tensor_slices(
(payload_test,host_test))
test_dataset=(test_dataset.map(normalize).batch(
1024,drop_remainder=True)).shuffle(500)

for payload,host in train_dataset:
train_host_psnr,train_payload_psnr,train_host_ssim,train_payload_ssim=train_step(
```

```
payload,host)

for payload,host in test_dataset:
test_host_psnr,test_payload_psnr,test_host_ssim,test_payload_ssim=test_step(
payload,host)

elapsed=time.time()-start
print('elapsed: %f' % elapsed)

template='Epoch {}, Train Loss: {}, Test Loss: {}, TrainH PSNR: {}, TrainP PSNR: {}, TestH
PSNR: {}, TestP PSNR: {}, TrainH SSIM: {}, TrainP SSIM: {}, TestH SSIM: {}, TestP
SSIM: {}'
print(
template.format(epoch+1,train_loss.result(),test_loss.result(),
train_host_psnr,train_payload_psnr,test_host_psnr,
test_payload_psnr,train_host_ssim,train_payload_ssim,
test_host_ssim,test_payload_ssim))

## append to metrics containers:
train_metrics.append([epoch+1,train_loss.result(),train_host_psnr,train_payload_psnr,train_hos
t_ssim,train_payload_ssim])
test_metrics.append([epoch+1,test_loss.result(),test_host_psnr,test_payload_psnr,test_host_ssi
m,test_payload_ssim])


# Reset the metrics for the next epoch
test_loss.reset_states()

## write the fcode to pipe output results into the summary directory


print('Training Finished.')
elapsed: 840.486465
Epoch 1, Train Loss: 0.8525092005729675, Test Loss: 0.6450870633125305, TrainH PSNR: 8
.0586576461792, TrainP PSNR: 9.852148056030273, TestH PSNR: 8.098840713500977, Test
P PSNR: 9.949888229370117, TrainH SSIM: 0.05837903916835785, TrainP SSIM: 0.293746
7098236084, TestH SSIM: 0.05898302420973778, TestP SSIM: 0.3018467426300049
elapsed: 858.553449
Epoch 2, Train Loss: 0.695637047290802, Test Loss: 0.4720149636268616, TrainH PSNR: 9.
403305053710938, TrainP PSNR: 12.94263744354248, TestH PSNR: 9.404909133911133, Te
stP PSNR: 13.177595138549805, TrainH SSIM: 0.07324186712503433, TrainP SSIM: 0.4436
410069465637, TestH SSIM: 0.0724235400557518, TestP SSIM: 0.4476797580718994
elapsed: 799.193974
Epoch 3, Train Loss: 0.6102049350738525, Test Loss: 0.4147225022315979, TrainH PSNR: 9
.986690521240234, TrainP PSNR: 14.058639526367188, TestH PSNR: 9.92824649810791, T
```

estP PSNR: 14.000663757324219, TrainH SSIM: 0.08866199851036072, TrainP SSIM: 0.488
7077212333679, TestH SSIM: 0.0886060819029808, TestP SSIM: 0.4897480607032776
elapsed: 830.895367
Epoch 4, Train Loss: 0.5566731691360474, Test Loss: 0.3805350065231323, TrainH PSNR: 1
0.36737060546875, TrainP PSNR: 14.261640548706055, TestH PSNR: 10.4735689163208, T
estP PSNR: 14.172012329101562, TrainH SSIM: 0.11017497628927231, TrainP SSIM: 0.497
958242892219, TestH SSIM: 0.11198648810386658, TestP SSIM: 0.4947769045829773
elapsed: 798.637399
Epoch 5, Train Loss: 0.5178820490837097, Test Loss: 0.34613215923309326, TrainH PSNR:
11.23253059387207, TrainP PSNR: 13.862574577331543, TestH PSNR: 11.34123802185058
6, TestP PSNR: 13.830072402954102, TrainH SSIM: 0.15962199866771698, TrainP SSIM: 0.
4789486527442932, TestH SSIM: 0.16111597418785095, TestP SSIM: 0.4768767356872558
6
elapsed: 766.606079
Epoch 6, Train Loss: 0.4868074059486389, Test Loss: 0.31860023736953735, TrainH PSNR:
11.94332504272461, TrainP PSNR: 13.959432601928711, TestH PSNR: 11.94505310058593
8, TestP PSNR: 14.042232513427734, TrainH SSIM: 0.2065558135509491, TrainP SSIM: 0.4
7229185700416565, TestH SSIM: 0.21143010258674622, TestP SSIM: 0.4758669734001159
7
elapsed: 768.428772
Epoch 7, Train Loss: 0.46083441376686096, Test Loss: 0.2933834493160248, TrainH PSNR:
12.639691352844238, TrainP PSNR: 14.262857437133789, TestH PSNR: 12.5304489135742
19, TestP PSNR: 14.233418464660645, TrainH SSIM: 0.2438051551580429, TrainP SSIM: 0.
4863579273223877, TestH SSIM: 0.24428924918174744, TestP SSIM: 0.4892815947532654
elapsed: 776.944519
Epoch 8, Train Loss: 0.43842363357543945, Test Loss: 0.27254578471183777, TrainH PSNR
: 13.06351089477539, TrainP PSNR: 14.714218139648438, TestH PSNR: 13.0438575744628
9, TestP PSNR: 14.780150413513184, TrainH SSIM: 0.27478548884391785, TrainP SSIM: 0.
5040328502655029, TestH SSIM: 0.28221869468688965, TestP SSIM: 0.5065077543258667
elapsed: 773.239911
Epoch 9, Train Loss: 0.4190461337566376, Test Loss: 0.25763222575187683, TrainH PSNR:
13.373844146728516, TrainP PSNR: 15.092974662780762, TestH PSNR: 13.3702468872070
31, TestP PSNR: 15.123798370361328, TrainH SSIM: 0.3039718270301819, TrainP SSIM: 0.
5266854763031006, TestH SSIM: 0.3065130114555359, TestP SSIM: 0.5319269299507141
elapsed: 762.328233
Epoch 10, Train Loss: 0.40226754546165466, Test Loss: 0.24612300097942352, TrainH PSN
R: 13.62101936340332, TrainP PSNR: 15.535749435424805, TestH PSNR: 13.527679443359
375, TestP PSNR: 15.519454956054688, TrainH SSIM: 0.327705442905426, TrainP SSIM: 0.
5540722608566284, TestH SSIM: 0.3270886540412903, TestP SSIM: 0.5606338977813721
elapsed: 769.091262
Epoch 11, Train Loss: 0.3876093626022339, Test Loss: 0.2374727427959442, TrainH PSNR:
13.919767379760742, TrainP PSNR: 15.80202579498291, TestH PSNR: 13.82372665405273
4, TestP PSNR: 15.818042755126953, TrainH SSIM: 0.34700000286102295, TrainP SSIM: 0.
5778602957725525, TestH SSIM: 0.34771865606307983, TestP SSIM: 0.5759410262107849
elapsed: 758.977894

Epoch 12, Train Loss: 0.37471139430999756, Test Loss: 0.23016254603862762, TrainH PSNR: 14.071939468383789, TrainP PSNR: 16.07119369506836, TestH PSNR: 14.17340087890625, TestP PSNR: 16.035985946655273, TrainH SSIM: 0.3662717640399933, TrainP SSIM: 0.593541145324707, TestH SSIM: 0.365613728761673, TestP SSIM: 0.5949463248252869
elapsed: 742.835897
Epoch 13, Train Loss: 0.3632451295852661, Test Loss: 0.22324636578559875, TrainH PSNR: 14.292581558227539, TrainP PSNR: 16.349403381347656, TestH PSNR: 14.262384414672852, TestP PSNR: 16.343488693237305, TrainH SSIM: 0.3807082772254944, TrainP SSIM: 0.6123675107955933, TestH SSIM: 0.38452860713005066, TestP SSIM: 0.608099102973938
elapsed: 747.670203
Epoch 14, Train Loss: 0.3529626727104187, Test Loss: 0.21747012436389923, TrainH PSNR: 14.552978515625, TrainP PSNR: 16.540019989013672, TestH PSNR: 14.576175689697266, TestP PSNR: 16.566360473632812, TrainH SSIM: 0.4014694094657898, TrainP SSIM: 0.6224476099014282, TestH SSIM: 0.40240830183029175, TestP SSIM: 0.619826078414917
elapsed: 710.911053
Epoch 15, Train Loss: 0.34366747736930847, Test Loss: 0.21204933524131775, TrainH PSNR: 14.776628494262695, TrainP PSNR: 16.68126106262207, TestH PSNR: 14.700521469116211, TestP PSNR: 16.765058517456055, TrainH SSIM: 0.415885329246521, TrainP SSIM: 0.6317976713180542, TestH SSIM: 0.4163203537464142, TestP SSIM: 0.6353222131729126
elapsed: 1368.666938
Epoch 16, Train Loss: 0.3352164030075073, Test Loss: 0.20705097913742065, TrainH PSNR: 14.895139694213867, TrainP PSNR: 16.900310516357422, TestH PSNR: 15.021305084228516, TestP PSNR: 16.931079864501953, TrainH SSIM: 0.4340832829475403, TrainP SSIM: 0.6426805257797241, TestH SSIM: 0.43580228090286255, TestP SSIM: 0.6400564908981323
elapsed: 726.702754
Epoch 17, Train Loss: 0.32748106122016907, Test Loss: 0.20226383209228516, TrainH PSNR: 15.07663631439209, TrainP PSNR: 17.031787872314453, TestH PSNR: 15.149849891662598, TestP PSNR: 17.062225341796875, TrainH SSIM: 0.444984495639801, TrainP SSIM: 0.6508725881576538, TestH SSIM: 0.44672131538391113, TestP SSIM: 0.6507658958435059
elapsed: 1290.660314
Epoch 18, Train Loss: 0.3203272521495819, Test Loss: 0.1959538459777832, TrainH PSNR: 15.339536666870117, TrainP PSNR: 17.140037536621094, TestH PSNR: 15.281736373901367, TestP PSNR: 17.21068572998047, TrainH SSIM: 0.4642629027366638, TrainP SSIM: 0.6545155048370361, TestH SSIM: 0.46644294261932373, TestP SSIM: 0.659400224685669
elapsed: 1074.335581
Epoch 19, Train Loss: 0.3133070170879364, Test Loss: 0.16997888684272766, TrainH PSNR: 15.447303771972656, TrainP PSNR: 17.212814331054688, TestH PSNR: 15.456987380981445, TestP PSNR: 17.200180053710938, TrainH SSIM: 0.47481822967529297, TrainP SSIM: 0.6593428254127502, TestH SSIM: 0.47032904624938965, TestP SSIM: 0.6605831384658813
elapsed: 1054.632869
Epoch 20, Train Loss: 0.30483105778694153, Test Loss: 0.12249027192592621, TrainH PSNR: 15.6844482421875, TrainP PSNR: 17.246898651123047, TestH PSNR: 15.735115051269531, TestP PSNR: 17.239612579345703, TrainH SSIM: 0.484903454780578 6, TrainP SSIM: 0.6623303890228271, TestH SSIM: 0.484829306602478, TestP SSIM: 0.6622839570045471
elapsed: 1129.134626

Epoch 21, Train Loss: 0.29568174481391907, Test Loss: 0.10525935888290405, TrainH PSN
R: 15.895872116088867, TrainP PSNR: 17.386898040771484, TestH PSNR: 15.86207389831
543, TestP PSNR: 17.397289276123047, TrainH SSIM: 0.5007910132408142, TrainP SSIM: 0
.6661981344223022, TestH SSIM: 0.5050360560417175, TestP SSIM: 0.6692792773246765
elapsed: 1148.829933
Epoch 22, Train Loss: 0.28686028718948364, Test Loss: 0.09864551573991776, TrainH PSN
R: 16.11905288696289, TrainP PSNR: 17.55894660949707, TestH PSNR: 16.2061843872070
3, TestP PSNR: 17.598844528198242, TrainH SSIM: 0.519779622554779, TrainP SSIM: 0.67
52214431762695, TestH SSIM: 0.5233446359634399, TestP SSIM: 0.67229163646698
elapsed: 1113.918126
Epoch 23, Train Loss: 0.2785791754722595, Test Loss: 0.09407495707273483, TrainH PSNR
: 16.37051773071289, TrainP PSNR: 17.69047737121582, TestH PSNR: 16.36600494384765
6, TestP PSNR: 17.745481491088867, TrainH SSIM: 0.5370417833328247, TrainP SSIM: 0.6
818613409996033, TestH SSIM: 0.5345796346664429, TestP SSIM: 0.6866289377212524
elapsed: 1328.814427
Epoch 24, Train Loss: 0.270837664604187, Test Loss: 0.09142178297042847, TrainH PSNR:
16.468090057373047, TrainP PSNR: 17.857725143432617, TestH PSNR: 16.5489406585693
36, TestP PSNR: 17.906147003173828, TrainH SSIM: 0.5497971177101135, TrainP SSIM: 0.
6876579523086548, TestH SSIM: 0.5507972240447998, TestP SSIM: 0.6860265731811523
elapsed: 2322.035207
Epoch 25, Train Loss: 0.2636071741580963, Test Loss: 0.08895526826381683, TrainH PSNR
: 16.556575775146484, TrainP PSNR: 17.939373016357422, TestH PSNR: 16.533861160278
32, TestP PSNR: 18.022953033447266, TrainH SSIM: 0.5571203231811523, TrainP SSIM: 0.
6928297281265259, TestH SSIM: 0.5566690564155579, TestP SSIM: 0.6962958574295044
elapsed: 1313.450253
Epoch 26, Train Loss: 0.25683775544166565, Test Loss: 0.08652383089065552, TrainH PSN
R: 16.706998825073242, TrainP PSNR: 18.14689826965332, TestH PSNR: 16.660472869873
047, TestP PSNR: 18.158857345581055, TrainH SSIM: 0.5710315704345703, TrainP SSIM: 0
.6973543167114258, TestH SSIM: 0.5659188032150269, TestP SSIM: 0.7007499933242798
elapsed: 1169.922880
Epoch 27, Train Loss: 0.25048890709877014, Test Loss: 0.08452799171209335, TrainH PSN
R: 16.759716033935547, TrainP PSNR: 18.268795013427734, TestH PSNR: 16.72225189208
9844, TestP PSNR: 18.26166534423828, TrainH SSIM: 0.5793609619140625, TrainP SSIM: 0
.7044007778167725, TestH SSIM: 0.5781359076499939, TestP SSIM: 0.7079145908355713
elapsed: 961.684079
Epoch 28, Train Loss: 0.24452240765094757, Test Loss: 0.08262735605239868, TrainH PSN
R: 16.869915008544922, TrainP PSNR: 18.438352584838867, TestH PSNR: 16.75517845153
8086, TestP PSNR: 18.336185455322266, TrainH SSIM: 0.5825610160827637, TrainP SSIM:
0.7101558446884155, TestH SSIM: 0.5830742120742798, TestP SSIM: 0.7106891870498657
elapsed: 855.514745
Epoch 29, Train Loss: 0.23890626430511475, Test Loss: 0.0808388739824295, TrainH PSNR
: 16.964323043823242, TrainP PSNR: 18.44672393798828, TestH PSNR: 16.9789848327636
72, TestP PSNR: 18.590389251708984, TrainH SSIM: 0.5933010578155518, TrainP SSIM: 0.
7140130996704102, TestH SSIM: 0.591705858707428, TestP SSIM: 0.7165195345878601
elapsed: 872.117777

Epoch 30, Train Loss: 0.23360788822174072, Test Loss: 0.07916802167892456, TrainH PSNR: 17.08384132385254, TrainP PSNR: 18.564815521240234, TestH PSNR: 17.094814300537
11, TestP PSNR: 18.5303955078125, TrainH SSIM: 0.5991453528404236, TrainP SSIM: 0.72
34472632408142, TestH SSIM: 0.6010589599609375, TestP SSIM: 0.7167081832885742
elapsed: 849.332760
Epoch 31, Train Loss: 0.2286023646593094, Test Loss: 0.07769855856895447, TrainH PSNR
: 17.111194610595703, TrainP PSNR: 18.73929214477539, TestH PSNR: 17.2035369873046
88, TestP PSNR: 18.77109146118164, TrainH SSIM: 0.606826901435852, TrainP SSIM: 0.72
4804699420929, TestH SSIM: 0.603663444519043, TestP SSIM: 0.7290921807289124
elapsed: 873.617142
Epoch 32, Train Loss: 0.22386281192302704, Test Loss: 0.0763203576207161, TrainH PSNR
: 17.232036590576172, TrainP PSNR: 18.761009216308594, TestH PSNR: 17.181575775146
484, TestP PSNR: 18.875301361083984, TrainH SSIM: 0.6110455989837646, TrainP SSIM: 0
.7274811267852783, TestH SSIM: 0.6098284721374512, TestP SSIM: 0.7285584211349487
elapsed: 840.949109
Epoch 33, Train Loss: 0.21936911344528198, Test Loss: 0.07505279034376144, TrainH PSN
R: 17.305179595947266, TrainP PSNR: 18.899940490722656, TestH PSNR: 17.37807655334
4727, TestP PSNR: 18.846044540405273, TrainH SSIM: 0.620002031326294, TrainP SSIM: 0
.7323130965232849, TestH SSIM: 0.6172023415565491, TestP SSIM: 0.733306884765625
elapsed: 804.660139
Epoch 34, Train Loss: 0.21510295569896698, Test Loss: 0.07386677712202072, TrainH PSN
R: 17.45333480834961, TrainP PSNR: 18.94664764404297, TestH PSNR: 17.4357128143310
55, TestP PSNR: 18.980867385864258, TrainH SSIM: 0.623530387878418, TrainP SSIM: 0.7
360258102416992, TestH SSIM: 0.6253141164779663, TestP SSIM: 0.735793948173523
elapsed: 802.481036
Epoch 35, Train Loss: 0.21104519069194794, Test Loss: 0.07262247800827026, TrainH PSN
R: 17.55000877380371, TrainP PSNR: 19.057788848876953, TestH PSNR: 17.322854995727
54, TestP PSNR: 18.953088760375977, TrainH SSIM: 0.6301088333129883, TrainP SSIM: 0.
7413991689682007, TestH SSIM: 0.6300567984580994, TestP SSIM: 0.7390472888946533
elapsed: 800.522731
Epoch 36, Train Loss: 0.2071816623210907, Test Loss: 0.0711933895945549, TrainH PSNR:
17.63498306274414, TrainP PSNR: 18.96841812133789, TestH PSNR: 17.480642318725586,
TestP PSNR: 19.14019012451172, TrainH SSIM: 0.6369296908378601, TrainP SSIM: 0.7405
444383621216, TestH SSIM: 0.6293586492538452, TestP SSIM: 0.7452515363693237
elapsed: 799.387058
Epoch 37, Train Loss: 0.20349784195423126, Test Loss: 0.07034813612699509, TrainH PSN
R: 17.57989501953125, TrainP PSNR: 19.128887176513672, TestH PSNR: 17.643083572387
695, TestP PSNR: 19.17596435546875, TrainH SSIM: 0.6381831169128418, TrainP SSIM: 0.
7491616010665894, TestH SSIM: 0.6388161182403564, TestP SSIM: 0.7494320869445801
elapsed: 796.517483
Epoch 38, Train Loss: 0.19997918605804443, Test Loss: 0.06951835751533508, TrainH PSN
R: 17.707351684570312, TrainP PSNR: 19.274757385253906, TestH PSNR: 17.57568740844
7266, TestP PSNR: 19.1827392578125, TrainH SSIM: 0.646403431892395, TrainP SSIM: 0.7
508841753005981, TestH SSIM: 0.6431014537811279, TestP SSIM: 0.7507889270782471
elapsed: 789.632606

Epoch 39, Train Loss: 0.19661661982536316, Test Loss: 0.06825809180736542, TrainH PSNR: 17.65829086303711, TrainP PSNR: 19.364215850830078, TestH PSNR: 17.720203399658203, TestP PSNR: 19.33892250061035, TrainH SSIM: 0.6519676446914673, TrainP SSIM: 0.75494384765625, TestH SSIM: 0.6464543342590332, TestP SSIM: 0.7570368051528931
elapsed: 790.493767
Epoch 40, Train Loss: 0.193398579955101, Test Loss: 0.06750307232141495, TrainH PSNR: 17.749248504638672, TrainP PSNR: 19.37795639038086, TestH PSNR: 17.897798538208008, TestP PSNR: 19.34320831298828, TrainH SSIM: 0.6558456420898438, TrainP SSIM: 0.7543679475784302, TestH SSIM: 0.6565346717834473, TestP SSIM: 0.7579227685928345
elapsed: 796.116877
Epoch 41, Train Loss: 0.19031469523906708, Test Loss: 0.06673382967710495, TrainH PSNR: 17.925552368164062, TrainP PSNR: 19.4178466796875, TestH PSNR: 17.808210372924805, TestP PSNR: 19.357013702392578, TrainH SSIM: 0.6562601327896118, TrainP SSIM: 0.7590948939323425, TestH SSIM: 0.6608814597129822, TestP SSIM: 0.7569671273231506
elapsed: 772.973650
Epoch 42, Train Loss: 0.1873576045036316, Test Loss: 0.0658930242061615, TrainH PSNR: 17.93622589111328, TrainP PSNR: 19.404857635498047, TestH PSNR: 17.935182571411133, TestP PSNR: 19.393596649169922, TrainH SSIM: 0.6647313833236694, TrainP SSIM: 0.7614234685897827, TestH SSIM: 0.6640127897262573, TestP SSIM: 0.7648262977600098
elapsed: 785.946234
Epoch 43, Train Loss: 0.18451939523220062, Test Loss: 0.06494689732789993, TrainH PSNR: 18.073665618896484, TrainP PSNR: 19.466951370239258, TestH PSNR: 17.956277847290004, TestP PSNR: 19.434892654418945, TrainH SSIM: 0.6687189340591431, TrainP SSIM: 0.7646055221557617, TestH SSIM: 0.6658512353897095, TestP SSIM: 0.761772096157074
elapsed: 790.580032
Epoch 44, Train Loss: 0.18179307878017426, Test Loss: 0.06435564160346985, TrainH PSNR: 18.160663604736328, TrainP PSNR: 19.557079315185547, TestH PSNR: 18.136676788330078, TestP PSNR: 19.664812088012695, TrainH SSIM: 0.6700564026832581, TrainP SSIM: 0.7696061134338379, TestH SSIM: 0.6725068688392639, TestP SSIM: 0.7680371999740601
elapsed: 790.969214
Epoch 45, Train Loss: 0.17917022109031677, Test Loss: 0.06364937126636505, TrainH PSNR: 18.087865829467773, TrainP PSNR: 19.649147033691406, TestH PSNR: 18.097343444822422, TestP PSNR: 19.529647827148438, TrainH SSIM: 0.6728516817092896, TrainP SSIM: 0.7724723815917969, TestH SSIM: 0.6728941202163696, TestP SSIM: 0.7711426019668579
elapsed: 774.006009
Epoch 46, Train Loss: 0.17664629220962524, Test Loss: 0.06278213858604431, TrainH PSNR: 18.069517135620117, TrainP PSNR: 19.6112003326416, TestH PSNR: 18.164154052734375, TestP PSNR: 19.663454055786133, TrainH SSIM: 0.67583829164505, TrainP SSIM: 0.7716816663742065, TestH SSIM: 0.6732292771339417, TestP SSIM: 0.7744969129562378
elapsed: 775.997766
Epoch 47, Train Loss: 0.1742148995399475, Test Loss: 0.06207086145877838, TrainH PSNR: 18.214706420898438, TrainP PSNR: 19.70984649658203, TestH PSNR: 18.107044219970703, TestP PSNR: 19.74557876586914, TrainH SSIM: 0.6798880100250244, TrainP SSIM: 0.7772781848907471, TestH SSIM: 0.6794137954711914, TestP SSIM: 0.7778133153915405
elapsed: 777.989839

Epoch 48, Train Loss: 0.17187194526195526, Test Loss: 0.06154779717326164, TrainH PSN
R: 18.182477951049805, TrainP PSNR: 19.798805236816406, TestH PSNR: 18.25357055664
0625, TestP PSNR: 19.654319763183594, TrainH SSIM: 0.6802441477775574, TrainP SSIM:
0.778293251991272, TestH SSIM: 0.6857994794845581, TestP SSIM: 0.7756301164627075
elapsed: 754.139924
Epoch 49, Train Loss: 0.16961072385311127, Test Loss: 0.06081220880150795, TrainH PSN
R: 18.31529998779297, TrainP PSNR: 19.805944442749023, TestH PSNR: 18.228548049926
758, TestP PSNR: 19.72165870666504, TrainH SSIM: 0.6863839626312256, TrainP SSIM: 0.
783734917640686, TestH SSIM: 0.6800760626792908, TestP SSIM: 0.7798665761947632
elapsed: 838.770530
Epoch 50, Train Loss: 0.1674281507730484, Test Loss: 0.0602848120033741, TrainH PSNR:
18.352745056152344, TrainP PSNR: 19.868871688842773, TestH PSNR: 18.3106231689453
12, TestP PSNR: 19.798124313354492, TrainH SSIM: 0.6882873177528381, TrainP SSIM: 0.
783988356590271, TestH SSIM: 0.6884980201721191, TestP SSIM: 0.7793628573417664
elapsed: 896.316913
Epoch 51, Train Loss: 0.16532009840011597, Test Loss: 0.05971582233905792, TrainH PSN
R: 18.28564453125, TrainP PSNR: 19.874914169311523, TestH PSNR: 18.36941909790039,
TestP PSNR: 19.96377182006836, TrainH SSIM: 0.6929464340209961, TrainP SSIM: 0.7850
245237350464, TestH SSIM: 0.690375804901123, TestP SSIM: 0.7885890007019043
elapsed: 799.784212
Epoch 52, Train Loss: 0.16328154504299164, Test Loss: 0.05906058847904205, TrainH PSN
R: 18.432666778564453, TrainP PSNR: 19.87587547302246, TestH PSNR: 18.365510940551
758, TestP PSNR: 19.851280212402344, TrainH SSIM: 0.6930536031723022, TrainP SSIM: 0
.7844885587692261, TestH SSIM: 0.6926990747451782, TestP SSIM: 0.7845296263694763
elapsed: 948.130298
Epoch 53, Train Loss: 0.16131016612052917, Test Loss: 0.058686546981334686, TrainH PS
NR: 18.440448760986328, TrainP PSNR: 19.95917320251465, TestH PSNR: 18.40374565124
5117, TestP PSNR: 19.999046325683594, TrainH SSIM: 0.6944164633750916, TrainP SSIM:
0.7894694805145264, TestH SSIM: 0.6956058144569397, TestP SSIM: 0.7877476215362549
elapsed: 927.647094
Epoch 54, Train Loss: 0.1594022810459137, Test Loss: 0.0581461600959301, TrainH PSNR:
18.47608184814453, TrainP PSNR: 20.018505096435547, TestH PSNR: 18.5107421875, Test
P PSNR: 19.93082046508789, TrainH SSIM: 0.6994068622589111, TrainP SSIM: 0.7919481
39667511, TestH SSIM: 0.7004708647727966, TestP SSIM: 0.787574052810669
elapsed: 974.722254
Epoch 55, Train Loss: 0.15755365788936615, Test Loss: 0.05766792967915535, TrainH PSN
R: 18.45941162109375, TrainP PSNR: 20.11455535888672, TestH PSNR: 18.5585746765136
72, TestP PSNR: 19.953903198242188, TrainH SSIM: 0.7002811431884766, TrainP SSIM: 0.
7955743074417114, TestH SSIM: 0.7039957642555237, TestP SSIM: 0.7936882972717285
elapsed: 807.231493
Epoch 56, Train Loss: 0.15576305985450745, Test Loss: 0.05708600953221321, TrainH PSN
R: 18.61795425415039, TrainP PSNR: 20.157852172851562, TestH PSNR: 18.527545928955
078, TestP PSNR: 19.986309051513672, TrainH SSIM: 0.7018132209777832, TrainP SSIM: 0
.7985563278198242, TestH SSIM: 0.7046107649803162, TestP SSIM: 0.7918421626091003
elapsed: 767.678141

Epoch 57, Train Loss: 0.1540268510580063, Test Loss: 0.05659114569425583, TrainH PSNR : 18.59682846069336, TrainP PSNR: 20.04100227355957, TestH PSNR: 18.68271255493164, TestP PSNR: 20.178993225097656, TrainH SSIM: 0.7052585482597351, TrainP SSIM: 0.796 1998581886292, TestH SSIM: 0.7078375816345215, TestP SSIM: 0.7991567254066467
elapsed: 749.926385
Epoch 58, Train Loss: 0.15234465897083282, Test Loss: 0.056566376239061356, TrainH PS NR: 18.61227798461914, TrainP PSNR: 20.11094856262207, TestH PSNR: 18.620258331298 828, TestP PSNR: 20.134510040283203, TrainH SSIM: 0.7078888416290283, TrainP SSIM: 0 .796933650970459, TestH SSIM: 0.7097254991531372, TestP SSIM: 0.7964146137237549
elapsed: 866.626831
Epoch 59, Train Loss: 0.15071099996566772, Test Loss: 0.055776722729206085, TrainH PS NR: 18.67562484741211, TrainP PSNR: 20.165302276611328, TestH PSNR: 18.62523651123 047, TestP PSNR: 20.105667114257812, TrainH SSIM: 0.7083578109741211, TrainP SSIM: 0 .8034709692001343, TestH SSIM: 0.7128587961196899, TestP SSIM: 0.8007375001907349
elapsed: 812.898725
Epoch 60, Train Loss: 0.1491241306066513, Test Loss: 0.055371370166540146, TrainH PSN R: 18.593793869018555, TrainP PSNR: 20.155441284179688, TestH PSNR: 18.65828514099 121, TestP PSNR: 20.266376495361328, TrainH SSIM: 0.7094749808311462, TrainP SSIM: 0 .8006744384765625, TestH SSIM: 0.7098060846328735, TestP SSIM: 0.8047167062759399
elapsed: 793.215461
Epoch 61, Train Loss: 0.1475832611322403, Test Loss: 0.05490012466907501, TrainH PSNR : 18.681137084960938, TrainP PSNR: 20.201202392578125, TestH PSNR: 18.811561584472 656, TestP PSNR: 20.172115325927734, TrainH SSIM: 0.7120552062988281, TrainP SSIM: 0 .804524838924408, TestH SSIM: 0.7157313823699951, TestP SSIM: 0.805476725101471
elapsed: 806.146958
Epoch 62, Train Loss: 0.14608526229858398, Test Loss: 0.05461941659450531, TrainH PSN R: 18.770362854003906, TrainP PSNR: 20.369869232177734, TestH PSNR: 18.66193389892 578, TestP PSNR: 20.22473907470703, TrainH SSIM: 0.7139090299606323, TrainP SSIM: 0. 8083384037017822, TestH SSIM: 0.7147945165634155, TestP SSIM: 0.8049270510673523
elapsed: 834.288251
Epoch 63, Train Loss: 0.14462998509407043, Test Loss: 0.05453791096806526, TrainH PSN R: 18.73031234741211, TrainP PSNR: 20.29785919189453, TestH PSNR: 18.8074951171875 , TestP PSNR: 20.15390968322754, TrainH SSIM: 0.71505206823349, TrainP SSIM: 0.81055 95111846924, TestH SSIM: 0.7154898643493652, TestP SSIM: 0.8078212738037109
elapsed: 828.684867
Epoch 64, Train Loss: 0.1432138830423355, Test Loss: 0.05389051139354706, TrainH PSNR : 18.78000259399414, TrainP PSNR: 20.36146354675293, TestH PSNR: 18.71826171875, Te stP PSNR: 20.240406036376953, TrainH SSIM: 0.7194153070449829, TrainP SSIM: 0.80827 56996154785, TestH SSIM: 0.7189577221870422, TestP SSIM: 0.8085813522338867
elapsed: 843.048916
Epoch 65, Train Loss: 0.14183533191680908, Test Loss: 0.053515560925006866, TrainH PS NR: 18.817537307739258, TrainP PSNR: 20.357332229614258, TestH PSNR: 18.7041397094 72656, TestP PSNR: 20.30046844482422, TrainH SSIM: 0.7180544137954712, TrainP SSIM: 0.8101745843887329, TestH SSIM: 0.7193296551704407, TestP SSIM: 0.8107171654701233
elapsed: 826.224038

Epoch 66, Train Loss: 0.14049261808395386, Test Loss: 0.053554728627204895, TrainH PS
NR: 18.833820343017578, TrainP PSNR: 20.36591148376465, TestH PSNR: 18.80255889892
578, TestP PSNR: 20.356693267822266, TrainH SSIM: 0.7238494753837585, TrainP SSIM: 0
.8091162443161011, TestH SSIM: 0.719595730304718, TestP SSIM: 0.8113861083984375
elapsed: 828.125489
Epoch 67, Train Loss: 0.1391870230436325, Test Loss: 0.05282272398471832, TrainH PSNR
: 18.865497589111328, TrainP PSNR: 20.47589111328125, TestH PSNR: 18.8178596496582
03, TestP PSNR: 20.618507385253906, TrainH SSIM: 0.7238242030143738, TrainP SSIM: 0.
8134185671806335, TestH SSIM: 0.719946026802063, TestP SSIM: 0.8197862505912781
elapsed: 835.004229
Epoch 68, Train Loss: 0.13791310787200928, Test Loss: 0.05250055715441704, TrainH PSN
R: 18.98550033569336, TrainP PSNR: 20.503732681274414, TestH PSNR: 18.813091278076
172, TestP PSNR: 20.47091293334961, TrainH SSIM: 0.72818922996521, TrainP SSIM: 0.81
50956630706787, TestH SSIM: 0.7218389511108398, TestP SSIM: 0.8168424963951111
elapsed: 841.803481
Epoch 69, Train Loss: 0.13667118549346924, Test Loss: 0.052036721259355545, TrainH PS
NR: 18.900299072265625, TrainP PSNR: 20.56008529663086, TestH PSNR: 18.95535087585
4492, TestP PSNR: 20.450908660888672, TrainH SSIM: 0.7251805067062378, TrainP SSIM:
0.8177604675292969, TestH SSIM: 0.7291407585144043, TestP SSIM: 0.8135503530502319
elapsed: 850.189091
Epoch 70, Train Loss: 0.13545991480350494, Test Loss: 0.05191097408533096, TrainH PSN
R: 18.973054885864258, TrainP PSNR: 20.514604568481445, TestH PSNR: 18.95592498779
297, TestP PSNR: 20.436813354492188, TrainH SSIM: 0.7298116683959961, TrainP SSIM: 0
.8135265111923218, TestH SSIM: 0.7261805534362793, TestP SSIM: 0.8150166869163513
elapsed: 823.353949
Epoch 71, Train Loss: 0.13427942991256714, Test Loss: 0.05164715647697449, TrainH PSN
R: 19.036428451538086, TrainP PSNR: 20.536060333251953, TestH PSNR: 19.04288482666
0156, TestP PSNR: 20.406068801879883, TrainH SSIM: 0.7302544713020325, TrainP SSIM:
0.820786714553833, TestH SSIM: 0.7339366674423218, TestP SSIM: 0.8183110356330872
elapsed: 843.697492
Epoch 72, Train Loss: 0.13312780857086182, Test Loss: 0.051312483847141266, TrainH PS
NR: 18.984920501708984, TrainP PSNR: 20.57611846923828, TestH PSNR: 18.96181106567
3828, TestP PSNR: 20.623151779174805, TrainH SSIM: 0.7321249842643738, TrainP SSIM:
0.8206920623779297, TestH SSIM: 0.7298680543899536, TestP SSIM: 0.8228846788406372
elapsed: 828.364222
Epoch 73, Train Loss: 0.13200263679027557, Test Loss: 0.050903353840112686, TrainH PS
NR: 19.045169830322266, TrainP PSNR: 20.61449432373047, TestH PSNR: 19.02956390380
8594, TestP PSNR: 20.449787139892578, TrainH SSIM: 0.731694221496582, TrainP SSIM: 0
.8215690851211548, TestH SSIM: 0.734054684638977, TestP SSIM: 0.8194772005081177
elapsed: 821.942503
Epoch 74, Train Loss: 0.13090428709983826, Test Loss: 0.05086936801671982, TrainH PSN
R: 18.99864959716797, TrainP PSNR: 20.59967041015625, TestH PSNR: 18.8941917419433
6, TestP PSNR: 20.66037368774414, TrainH SSIM: 0.7335208654403687, TrainP SSIM: 0.82
46142268180847, TestH SSIM: 0.732691764831543, TestP SSIM: 0.8212451934814453
elapsed: 827.351195

Epoch 75, Train Loss: 0.12983256578445435, Test Loss: 0.05052118003368378, TrainH PSN
R: 18.99050521850586, TrainP PSNR: 20.584625244140625, TestH PSNR: 18.976730346679
688, TestP PSNR: 20.681964874267578, TrainH SSIM: 0.7322854399681091, TrainP SSIM: 0
.8228892087936401, TestH SSIM: 0.7324180603027344, TestP SSIM: 0.8260979056358337
elapsed: 840.313060
Epoch 76, Train Loss: 0.12878459692001343, Test Loss: 0.050149064511060715, TrainH PS
NR: 19.003890991210938, TrainP PSNR: 20.719989776611328, TestH PSNR: 19.1432266235
35156, TestP PSNR: 20.657752990722656, TrainH SSIM: 0.7336587905883789, TrainP SSIM
: 0.8269144296646118, TestH SSIM: 0.7388920783996582, TestP SSIM: 0.825795888900756
8
elapsed: 1137.375865
Epoch 77, Train Loss: 0.12776073813438416, Test Loss: 0.050009001046419144, TrainH PS
NR: 19.11092185974121, TrainP PSNR: 20.681446075439453, TestH PSNR: 19.13266181945
8008, TestP PSNR: 20.675426483154297, TrainH SSIM: 0.7372409105300903, TrainP SSIM:
0.8250179886817932, TestH SSIM: 0.7383098602294922, TestP SSIM: 0.8289744853973389
elapsed: 1201.547739
Epoch 78, Train Loss: 0.1267613172531128, Test Loss: 0.04948483407497406, TrainH PSNR
: 19.179122924804688, TrainP PSNR: 20.790796279907227, TestH PSNR: 19.131790161132
812, TestP PSNR: 20.747026443481445, TrainH SSIM: 0.7398650646209717, TrainP SSIM: 0
.828311562538147, TestH SSIM: 0.7420396208763123, TestP SSIM: 0.8271902799606323
elapsed: 828.017638
Epoch 79, Train Loss: 0.1257820576429367, Test Loss: 0.04928835108876228, TrainH PSNR
: 19.121562957763672, TrainP PSNR: 20.915496826171875, TestH PSNR: 19.177324295043
945, TestP PSNR: 20.746414184570312, TrainH SSIM: 0.7367548942565918, TrainP SSIM: 0
.83073890209198, TestH SSIM: 0.7426016330718994, TestP SSIM: 0.8271912336349487
elapsed: 865.732117
Epoch 80, Train Loss: 0.12482418119907379, Test Loss: 0.04924488440155983, TrainH PSN
R: 19.145404815673828, TrainP PSNR: 20.854461669921875, TestH PSNR: 19.08421707153
3203, TestP PSNR: 20.7666015625, TrainH SSIM: 0.7393167018890381, TrainP SSIM: 0.832
7029943466187, TestH SSIM: 0.7363786101341248, TestP SSIM: 0.8307434320449829
Training Finished.

In [34]:

```
steganography_model.save_weights('./model_weights/weights.h5')
# Save the weights of the model
```

In [18]:

```
train_df=pd.DataFrame(train_metrics,columns=['Epoch','Train Loss','TrainH PSNR','TrainP
PSNR','TrainH SSIM','TrainP SSIM'])
test_df=pd.DataFrame(test_metrics,columns=['Epoch','Test Loss','TestH PSNR','TestP
PSNR','TestH SSIM','TestP SSIM'])
```

In [21]:

```
from openpyxl.workbook import Workbook
```

In [22]:

```
excel_file_path='training_results.xlsx'
with pd.ExcelWriter(excel_file_path) as writer:
train_df.to_excel(writer,sheet_name='Train Metrics',index=False)
test_df.to_excel(writer,sheet_name='Test Metrics',index=False)
```

In [23]:

```
payload_test=x_test[np.random.choice(np.arange(x_test.shape[0]),
size=x_test.shape[0])]
if payload_image_shape[-1]==1:
# for when payload is grayscale
# for when payload is grayscale
payload_test=np.expand_dims(np.mean(payload_test,axis=-1),axis=-1)

host_test=x_test[np.random.choice(np.arange(x_test.shape[0]),
size=x_test.shape[0])]
# Instantiate the test Dataset class
test_dataset=tf.data.Dataset.from_tensor_slices((payload_test,host_test))
test_dataset=(test_dataset.map(normalize).batch(
256,drop_remainder=True)).shuffle(500)

for payload,host in test_dataset:
test_host_psnr,test_payload_psnr,test_host_ssim,test_payload_ssim=test_step(
payload,host)

print("Test Loss: ",test_loss.result(),"PSNR-H: ",test_host_psnr,
"PSNR-P: ",test_payload_psnr)
```

```
Test Loss:  tf.Tensor(0.04876513, shape=(), dtype=float32) PSNR-H:  tf.Tensor(19.17472, sha
pe=(), dtype=float32) PSNR-P:  tf.Tensor(20.713905, shape=(), dtype=float32)
```

In [24]:

```
def test_normalize(payload,host):
normalized_payload=tf.divide(
tf.math.subtract(payload,tf.reduce_min(payload)),
tf.math.subtract(tf.reduce_max(payload),tf.reduce_min(payload)))
normalized_host=tf.divide(
tf.math.subtract(host,tf.reduce_min(host)),
tf.math.subtract(tf.reduce_max(host),tf.reduce_min(host)))
return normalized_payload,normalized_host,payload,host
```

In [ ]:

Evaluation

In [25]:

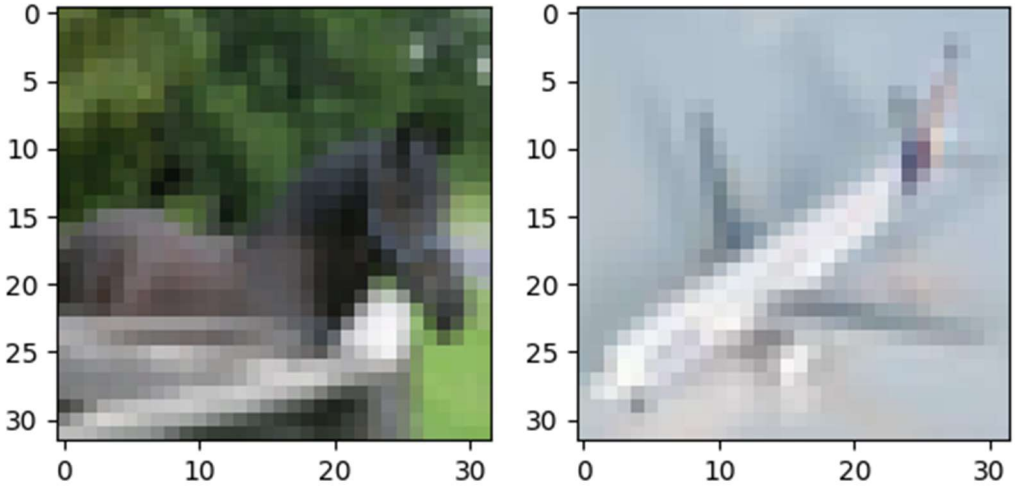
```
def normed_to_image(normed_image,min_,max_):
return (normed_image*(max_-min_))+min_
```

In [26]:

```
example_ids=np.arange(len(host_test))[:100]
example_id=np.random.choice(example_ids)

# showing host
fig,axs=plt.subplots(ncols=2)

host_example=host_test.astype(int)[example_id]
payload_example=payload_test.astype(int)[example_id]

# payload_example = np.concatenate(
#     (payload_example, payload_example, payload_example), axis=-1)
axs[0].imshow(host_example)
axs[1].imshow(payload_example,cmap='gray')
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x20339333f08>
```



In [27]:

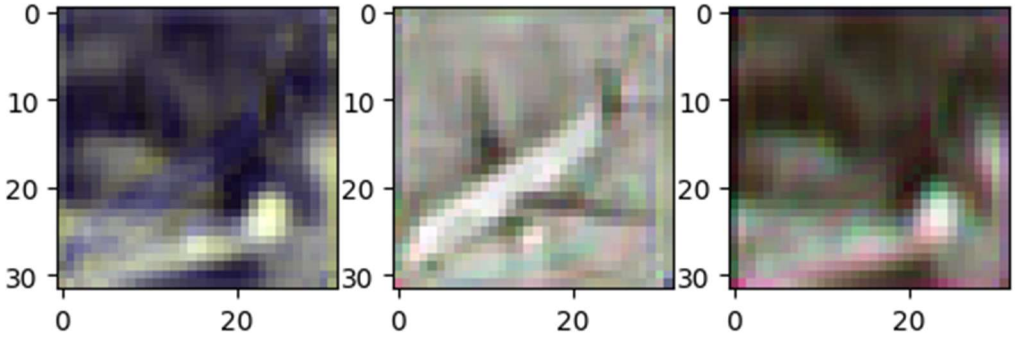
```
# showing host
fig,axs=plt.subplots(ncols=3)

inference_dataset=tf.data.Dataset.from_tensor_slices(
(payload_test[example_ids],
host_test[example_ids])).map(test_normalize).batch(len(example_ids))
for norm_payload,norm_host,payload,host in inference_dataset:
encoded_host,decoded_payload,decoded_host=steganography_model(
[norm_host,norm_payload])
host_output=encoded_host.numpy()
payload_output=decoded_payload.numpy()
decoded_host_output=decoded_host.numpy()

host_output=normed_to_image(host_output,np.min(host),np.max(host))
```

```
payload_output=normed_to_image(payload_output,np.min(payload),
np.max(payload))
decoded_host_output=normed_to_image(decoded_host_output,np.min(host),
np.max(host))

host_output=host_output.astype(int)[example_id]
payload_output=payload_output.astype(int)[example_id]
decoded_host_output=decoded_host_output.astype(int)[example_id]

# payload_output = np.concatenate(
#     (payload_output, payload_output, payload_output), axis=-1)

axs[0].imshow(host_output)
axs[1].imshow(payload_output)
axs[2].imshow(decoded_host_output)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] fo
r integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] fo
r integers).
```

Out[27]:

```
<matplotlib.image.AxesImage at 0x2033a233808>
```



In [ ]: