International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Deep Steganography using CNN and Machine Learning Techniques

Poonam Yadav[1], Ashka Bhalodia[2]

[1]ME (Computer Engineering),
[2]Computer Engineering Department,
[1,2]Bhagwan Mahavir College of Engineering & Technology, Bhagwan Mahavir University, Surat, Gujarat, India

Abstract

This paper will seek to look at steganography where techniques to be embraced will use images in bid to encase secret data within other files that appear harmless. The largest motivation will come from steganography's novelty feature and high security which would also safeguard information from future intruders. Thus, a complex deep steganography model will be Developed by implementing the Machine Learning and Steganography approaches. The purpose of the model will be to embed one image into another in order that cannot easily be distinguished and at the same time the quality of the images will not be distorted greatly.

As such, the main objective of this study will be to optimise the Editors' ability to balance the potential of information embedding with the image quality in order to address current challenges in the field of information security. With the help of solution with machine learning methods, the model will be able to change the payload depending on the characteristics of the cover image. This will enhance the fortification of the model guarding temporary sensitive data in the digital terrain by strengthening and diversifying it.

The outcome of this research will therefore be a highly developed deep steganography model that will be integrated to specifically hide and extract images undetectably. The model will be trained by CNNs and RNNs, respectively, in an iteratively fashion. Specific loss functions will be employed in order to prevent excessive distortion of the cover pictures and to optimize the ability to hide data.

The results will be analyzed and discussed in terms of quantitative measures including PSNR, SSIM and the message recovery rate and general visual inspections. The additional validations, such as the robustness tests, computational efficiency assessments, and the security tests will increase confidence in the model's performance as well as in its reliability.

Consequently, the deep steganography model presented in this paper will utilize modern forms of learning as well as steganographic skills to form a considerable strategy in concealing and recovering sensitive data in images. The sequential process of establishing and testing the model will demonstrate its applicability for use in safeguarding digital information in an increasingly integrated global environment.

Keywords: Deep Steganography, Convolutional Neural, Networks (CNNs),Image Embedding, Information Security, Machine Learning

I. INTRODUCTION

The hidden message technique known as steganography developed from its beginning during 440 BCE through extensive progress since then. During its original stage it incorporated physical message-hiding techniques including the practice of writing on wax tablets and the use of invisible inks. Through centuries of development this concealed communication form welcomed technological progress that now leads to digital use for information security against unauthorized access and cybersecurity dangers.

The word "steganography" comes from two Greek elements which combine the words steganos (hidden) and graphie (writing) to create its meaning. According to its original meaning steganography represents a method to insert covert messages inside different communications so they remain undetectable. Steganography exists in the modern digital world because it relies on advanced methods which protect valuable information against cybersecurity dangers and hacking attempts.

Image steganography stands out as the most widely used digital steganography type because images offer both extensive capacity for data storage and plentiful availability in digital systems. The method of image steganography uses pixel value adjustments to a “host image” for secret data encryption purposes. The embedding process has two main technique categories which include spatial domain approaches and transform domain approaches.

The Spatial domain technique uses LSB manipulation to modify image pixel values for information embedding purposes. The two transform domain methods DCT and DWT function on separate frequency domains within the image. The security of embedded data is improved through transform domain methods which create data that resists typical image modifications such as compression and scaling.

Deep learning techniques have revolutionized steganography research through their application during the previous several years. Neural networks enabled better methods and optimization of hidden data processing in recent years. Deeper learning methods use algorithmic abilities to discover optimal methods for conducting steganographic operations. Image steganography benefits from Convolutional Neural Networks (CNNs) because these networks easily handle pixel-level features. These modern neural network models surpass previous detection systems through their adaptive learning frameworks and enhanced design which produces better perceptibility and robustness.

The persistent evolution in this field produced improved steganographic quality and established new feasible possibilities. Latest neural network technology enables models to maintain image visual quality while securing hidden information in an unobservable way.

The goal of this research is to construct an improved steganalytic model which solves current technical constraints based on recent advancements in deep learning steganography. Future steganographic models will build on this work to achieve better accuracy and better resilience against modern challenges while keeping their embedded security robust. The research effort seeks to link modern steganographic performance with forthcoming advancements while sustaining the evolution of this field during the digital period.

II. OBJECTIVE

The main research purpose and boundaries of this project focus on creating superior deep steganography methods that use neural networks to make images more secure. The development targets an efficient system for concealing and retrieving secret images from cover images utilizing methods which achieve security along with imperceptibility and adaptability to various circumstances. This research addresses modern information technology requirements by developing a deep stegano network which focuses on operational detectability as well as system resilience and adaptability.

The development requires a dual network architecture consisting of two neural networks.

- ✓ The design features a hiding neural network that embeds secret images while maintaining cover image visual quality during the secret image container development.
- ✓ The solution needs both algorithms to extract hidden images precisely from network containers in a way that protects their original condition.

The system must find an ideal point where embedding capabilities meet imperceptibility needs.

- ✓ The process will create container images with superior visual quality to hide secret data that cannot be discovered by human perception or complex steganalysis programs.
- ✓ The introduction of new loss functions during deep learning operates to reduce distortions between cover and secret images for optimal embedding results.

Incorporate dynamic payload adjustment:

- ✓ A machine learning system should adjust the hidden data allocation proportionally to the cover image density and content type through automated algorithms.
- ✓ The technique enables compatibility with cover images featuring different resolution settings along with varying format and content types thus making it operational in diverse application circumstances.

The method needs to become resistant to both steganalysis detection and image processing attacks.

- ✓ The methodology should demonstrate resistance to typical image processing attacks which include compression as well as resizing and addition of noise to the images.
- ✓ The method must maintain resistance to all statistical steganalysis programs to qualify as a practical solution for real-world conditions.

This investigation seeks to create an essential framework that supports safe and dependable adaptive steganographic approaches. The research results will unlock applications for secure communication along with information confidentiality and piracy prevention within extensive domains. A robust system which enables security, functionality and low-detectability will become possible through this approach for future innovation in steganography.

III. PROBLEM STATEMENT

Limitations of Traditional Steganography Techniques

The conventional steganography techniques though classical in approach poses severe drawbacks in the present world. The two major restrictions are the restricted amount of data that can be concealed and the inability to ensure absolute protection for the embedded data. The effectiveness of these techniques declines in protecting data because present-day adversarial technologies and advanced digital forensic tools make the data more exposed to unauthorized access. The limited payload capacity acts as a major obstacle for using this method to hide large amounts of data. This self-imposed need requires changing to alternative protection techniques that defeat current constraints to meet growing demands within the new technological era.

The Growing Demand for Advanced Steganographic Solutions

Security threats against systems and information continue to increase in complexity thus creating sufficient reason to develop better steganography methods. Modern steganographic methods fail to shield hidden information from emerging adversarial threats together with heuristic-based detection methods which motivates researchers to enhance steganographic resilience. An increasing demand requires the establishment of advanced techniques based on deep learning frameworks to address this necessity. Using the adaptive along with predictive features of deep neural networks these new methods enable developers to create new measures which boost security robustness and operational efficiency throughout steganography applications.

Objectives for Deep Learning-Driven Steganography Models

The goal of this research involves creating and deploying an original steganography model that utilizes deep learning models as the primary infrastructure. The proposed system strives to maximize its stealth capacity while minimizing the host image deterioration while also maximizing hidden data volume. The proposed solution addresses critical standard approach weaknesses by giving both stealth capabilities and reliability control. This work seeks to boost the protection level against detection and extraction attacks across a wide range of tasks and threats affecting the hidden data within the model.

Challenges in Algorithm Optimization and Robustness

Several technical problems arise from the process of generating optimal embedding algorithms because the task requires hiding images while preserving essential elements of the cover image at a nonvisible level. Two critical aspects exist equally in line with each other; one is improving concealment efficiency during large-object embedding and the other is maintaining content integrity. Constant research focuses on enhancing the model's resistance to adversarial image-based threats that include compression methods and malignant distortions together with noise attacks. These proposed solutions need the achievement of two essential goals that combine reliable high-security systems with minimum-distorted market conditions to enable real-world implementation of these principles.

Towards a Practical and User-Friendly Steganography Tool

Research ensures the proposed steganographic solution achieves practical usability by addressing theoretical needs for deployability. The planned software application supports users with or without previous experience in steganographic techniques to operate it successfully. This proposed deep learning based method integrated with user-friendly capabilities provides people and organizations with real access to secure data embedding and extraction so they can use advanced steganographic solutions for diverse practical needs.

IV. LITERATURE REVIEW

Recent developments in steganalysis and steganography have increasingly utilized deep learning methods, particularly Convolutional Neural Networks (CNNs), to improve performance and address the limitations of conventional methods. Ntivuguruzwa et al. (2024) showed how CNNs can efficiently detect anomalies in stego-images by automatically learning spatial features and combining classification and feature extraction. Kumar et al. (2020) also highlighted the advantage of CNNs over conventional classifiers due to their self-learning ability, allowing accurate identification even at low embedding ratios. Other research, such as by Hashemi et al. (2022), introduced models with convolutional autoencoders and ResNet to achieve high imperceptibility with PSNR of more than 40 dB and SSIM of more than 0.98 and high resistance to steganalysis attacks. Kumar et al. (2020) also introduced a dual-network framework with H-net and R-net and achieved robust embedding and retrieval with low distortion. Utilization of hybrid activation functions, adaptive payload control, and loss functions tailored for image quality further improved steganographic performance. The models showed their robustness on a range of datasets such as BOSSBase, COCO, and CelebA. In summary, the combination of deep learning with steganography has transformed the field, allowing the construction of flexible, secure, and high-capacity systems for real-world applications in sensitive communication contexts.

Comparative study

No.	Study	Focus	Methodology	Key Findings
1	Ntivuguruzwa, Ahmad, & Han (2024)	The efficiency of DL-based steganalysis experiences difficulties in detecting concealed content	The architecture combines high-pass filters with multiple levels of deep learning along with ReLU/Sigmoid activation functions under adaptive learning models	The system achieved enhanced accuracy when detecting images and JPEG files in addition to solving challenges related to classifier size and adaptive learning procedures for solid detection capabilities.
2	Kumar, Rao, & Choudhary (2020)	The research showed better spatial and JPEG detection accuracy while improving classifier dimensions alongside adaptive learning methods.	The system achieved better accuracy results in spatial and JPEG testing fields while improving learning methods to enhance operational detection capabilities.	High accuracy levels were achieved at low embedding rates and it displayed better results in detecting multiple steganography classes through its adaptable and speedy learning system..
3	Kumar, Laddha, Sharma, & Dogra (2020)	Image steganography using LSB and CNNs	H-net and R-net optimized using Adam algorithm	Achieved low distortion, high security, and compatibility across various image types; suitable for secure communication, watermarking, and forensic analysis.
4	Hashemi,	Improved color	Convolutional	High imperceptibility

	Majidi, &Khorashadizadeh (2022)	image steganography using autoencoders and ResNet	autoencoders for feature extraction and ResNet for embedding/extraction	(PSNR > 40 dB, SSIM > 0.98); robust against steganalysis attacks; high capacity (8 bits/pixel) for secure communication and digital watermarking.
5	Laxmi & Rajkumar (2023)	Systematic review of classical and DL-based image steganography	Analysis of transform and spatial domain techniques, adversarial examples, and GANs	Highlighted deep learning's role in embedding robustness and security; advocated using advanced neural structures like GANs for enhanced steganographic capabilities.
6	Kaneria&Jotwani (2024)	Comparative analysis of deep learning encoders (U-Net, V-Net, U-Net++)	U-Net for high-quality embedding; comparison based on PSNR, MAE, and VIF	U-Net achieved superior embedding capacity and quality compared to other encoders; suitable for flexible and reliable steganographic systems.
7	Płachta, Rudziński, &Śmigielski (2022)	Detecting steganographic content in JPEG images	Sampling with ensemble classifiers and linear regression on DCTR and GFR features	Ensemble classifiers outperformed deep learning in detecting high embedding rates; provided insights into optimizing security systems against stegomalware.
8	Yola et al. (2023)	Enhancing CNN-based steganalysis with hybrid activation functions	Hybrid activation functions to improve feature extraction and classification	Achieved 81% accuracy; hybrid activations outperformed traditional functions; provided strong foundational work for secure communication and image manipulation.
9	Bhatt, Patel, & Shah (2024)	Deep steganography model using CNNs	Smart encoder-decoder structures with convolutional filters and novel loss functions	Enabled dynamic payload steganography with low distortion; demonstrated improved capacity, visualization, and security through practical software implementation.
10	Hegarty & Keane (2020)	Advanced steganography detection using CNNs	Adjusted CNN parameters (filters, epochs, activation functions)	Achieved higher accuracy and minimized false positives; applicable in detecting covert

				communication and industrial espionage in cybersecurity.
11	Ntivuguruzwa& Ahmad (2023)	Spatial DL-based steganalysis	Depthwise separable CNN, multi-scale pooling, LReLU	Improved detection by 10.2%; robust across image sizes
12	Xie, Ren et al. (2019)	Traditional vs DL-based steganalysis	Survey of SPAM, SRM, DCTR vs CNNs	DL (e.g., SRNet) surpasses classical methods; calls for synergy
13	Kumar, Laddha, Aniket & Dogra (2021)	Image & text steganography	Dual CNN with LSB, masking, line shift, Adam optimizer	Strong concealment, text/image support, no retraining needed
14	Himthani et al. (2022)	Performance comparison of DL encoders	U-Net, V-Net, U-Net++ + unified decoder	U-Net best in MSE, PSNR, SSIM; practical for secure embedding
15	Guzman (2022, thesis)	Optimized DL steganography	CNNs with gain function (PSNR + SSIM), no normalization	High fidelity (SSIM 0.9971, PSNR 43.04 dB); effective for various image types
16	Khalifa & Guzman (2022)	Symmetry-based image steganography	SteGuz model with 3 CNNs, gain function, no noise layer	Superior imperceptibility and extraction; improved over baseline

V. METHODOLOGY

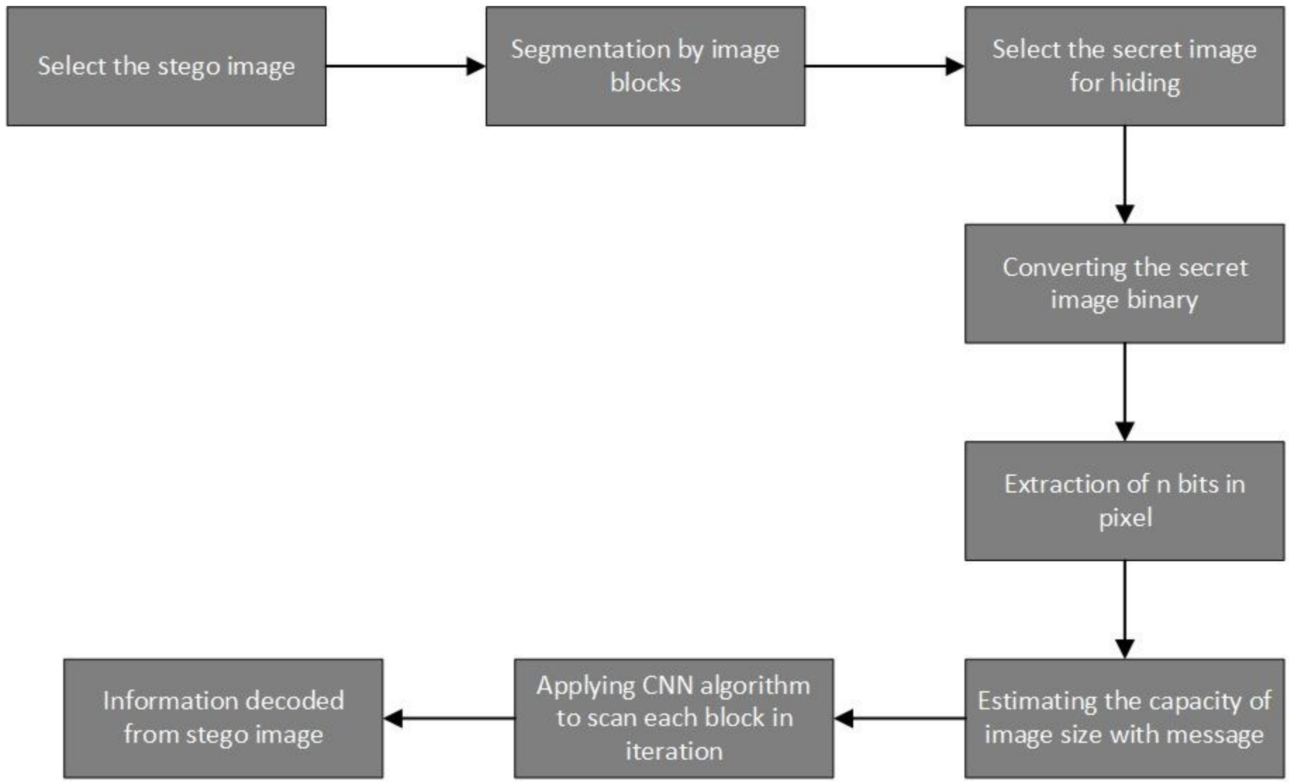


Fig 1: Proposed Flow of the Model

1. Data Collection and Preparation

The following steps create a steady base for both training and evaluation procedures:

A diverse dataset which includes both cover and secret images is obtained for the process. Multiple types of images with varied content are specifically chosen as cover assets for the dataset to maintain heterogeneous representation.

A preprocessing step involves the simultaneous treatment of secret images which contain information for hiding together with cover images. Preprocessing steps include:

Resizing: Adjusting dimensions to a uniform resolution for compatibility.

The training process requires normalization of pixel values which achieves standardization that helps training to succeed.

2. Model Selection

A system must reach its peak performance for processing image-based steganographic content by following these factors:

CNNs serve as the main architecture selection because of their established capability in extracting image features and recognizing patterns.

The recurrent neural networks (RNNs) demonstrate capability to process sequences of data because they assist with the management of sequential dependencies found in steganographic operations.

3. Training Process

The training method applies multiple repetitions to improve capacity for secret image insertion into cover images effectively:

A systematic adjustment of model parameters leads to optimized embedding capacity during the parameter optimization process.

A specialized loss function gets designed so that the following criteria achieve balance:

- Minimizing distortion in the cover images.
- Ensuring effective concealment of secret information.

Stego-image creation focuses on making images with superior quality that show no detectable changes when observed against their original cover versions.

4. Embedding Algorithm Optimization

The main focus lies on embedding algorithm optimization for increased performance effectiveness.

The optimization of embedding procedures is carried out by implementing regularization techniques alongside hyperparameter adjustment methods for improved efficiency.

The algorithms undergo systematic modifications for minimizing visible distortions in cover images without compromising the integrity of hidden data.

5. Robustness Assessment

The model faces extensive testing during evaluations of its ability to withstand different adversarial threats:

Attack Simulations it is important to subject the model to resizing and compression processes as well as steganalysis testing.

The system tests its performance through robustness metrics which verify secret data protection when operating across different scenarios that include environmental changes and computational conditions.

6. User-Friendly Software Tool Development

A user-friendly software solution for steganographic purposes has been created because of its real-world significance.

The system uses an interface that fits seamlessly into user operations to enable effortless hiding and extracting of secret images from cover files.

Usability Testing: A broad testing process verifies that experts and non-experts alike can utilize the tool for steganography purposes to bridge practical steganography development with real-usage needs.

7. Performance Evaluation and Comparison

Performance examination of the model utilizes quantitative together with qualitative assessment procedures.

Security metrics and visual quality and embedding capacity are the metrics chosen for assessment.

A benchmarking process establishes relationships between current approaches in steganography to demonstrate how the proposed strategy surpasses others by using improved efficiency, robustness and covert integration capabilities.

8. Documentation and Reporting

The research needs full disclosure and repeatability through the following measures:

The research preserves complete documentation which records experimental methods together with selected parameters and produced results.

The final reports include a summary of findings and show how the research advances deep steganography with machine learning and what additional improvements can follow.

VI. IMPLEMENTATION

The artistic paintings contained in Tiny ImageNet's distribution are small vivid pictures sized 64x64 pixels. The distribution canvas available at (https://www.kaggle.com/ datasets /nikhilshingadiya/tinyimagenet200) contains 200 specific classes that establish their own rules.

The procedure in training incorporates 500 images from each category that builds a refined artistic dataset for digital preparations. The next step in the procedure allows classifiers to evaluate artworks that comprise 50 pictures per category for validation purposes. The evaluation phase reveals all testing photos in their complete sequence of 50 images per class as the grand finale to determine artistry performance.

A. Data Pre-Processing

The first part of the code defines required files belonging to both training and testing sets within the Tiny ImageNet dataset after specifying needed libraries.

```
!pip install kaggle

Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.12.14)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.10)

from google.colab import files
files.upload()

Choose Files kaggle.json
• kaggle.json(application/json) - 70 bytes, last modified: 1/5/2025 - 100% done
Saving kaggle.json to kaggle.json
```

X_train and x_test arrays are created through data preparation using numpy arrays which contain 2,000 pictures among which 1,000 will be trained and the remaining 1,000 serve analysis purposes.

The training dataset (tiny-imagenet-200/train) receives seven iterative loops for random insertion of ten images per category to create x_train. The training data becomes diverse containing multiple category pictures because of this method implementation.

The pixel value normalization process begins after the image laden algorithm completely loads training and test pictures. The preprocessing technique of normalisation represents a common machine learning method through which all input variable ranges become standardised. The patient’s pixel values become stored within the interval [0,1] after dividing each value by 255.0.

```
files = os.listdir('tiny-imagenet-200/train')
files_te = os.listdir('tiny-imagenet-200/test/images')


x_train = np.empty((2000,64,64,3), 'uint64')
a=0
for i in range(200):
  idd = np.random.randint(0, 500, 10)
  for j in range(10):
    image = cv2.imread('tiny-imagenet-200/train/'+files[i]+'/images/'+files[i]+'_'+str(idd[j])+'.JPEG')
    x_train[a] = image
    a=a+1

x_test = np.empty((2000,64,64,3), 'uint64')
a=0
for i in range(2000):
    image = cv2.imread('tiny-imagenet-200/test/images/'+files_te[i])
    x_test[a] = image
    a=a+1

input_S = x_train[0:1000]

input_C = x_train[1000:]

input_C = input_C/255.0
input_S = input_S/255.0
```

Fig 2: Pseudocode for the Data Pre-Processing

In Data Division input_S contains the first thousand images and input_C represents the second set of thousand images after total normalization of training images. The authors have probably divided the data into these datasets to fulfillsteganographical purposes since input_S represents messages that need hiding and input_C stands for the hiding components.

The compatibility of next operations depends on changing input_C and input_S data types to float64. The goal of this phase consists in ensuring uniformity of data types throughout the entire script.

Deep steganography plays a vital role within this code section because it handles all picture data preparation tasks. The optimization phase includes importing photos together with image segmentation for training and testing purposes and pixel intensity adjustment for standardization and steganography data preparation duties.

```
array([[[[0.77254902, 0.7254902 , 0.70196078],
         [0.78039216, 0.73333333, 0.70980392],
         [0.79215686, 0.74509804, 0.72156863],
         ...,
         [0.79215686, 0.73333333, 0.72156863],
         [0.77254902, 0.70980392, 0.68627451],
         [0.75294118, 0.69019608, 0.66666667]],

        [[0.77254902, 0.7254902 , 0.70196078],
         [0.78039216, 0.73333333, 0.70980392],
         [0.79215686, 0.74509804, 0.72156863],
         ...,
         [0.78823529, 0.72941176, 0.71764706],
         [0.77254902, 0.70980392, 0.68627451],
         [0.75686275, 0.69411765, 0.67058824]],

        [[0.78431373, 0.72941176, 0.70588235],
         [0.79215686, 0.7372549 , 0.71372549],
         [0.8       , 0.74509804, 0.72156863],
         ...,
         [0.78431373, 0.7254902 , 0.71372549],
         [0.77647059, 0.71372549, 0.69019608],
         [0.76470588, 0.70196078, 0.67843137]],

        [[0.27843137, 0.34117647, 0.36470588],
         [0.3254902 , 0.38823529, 0.41176471],
         [0.27058824, 0.33333333, 0.35686275],
         ...,
         [0.30196078, 0.36470588, 0.41176471],
         [0.36862745, 0.43137255, 0.47843137],
         [0.40392157, 0.46666667, 0.51372549]],

        [[0.20784314, 0.28235294, 0.30980392],
         [0.2       , 0.2745098 , 0.30196078],
         [0.24705882, 0.32156863, 0.34901961],
         ...,
         [0.31372549, 0.38823529, 0.41960784],
         [0.43137255, 0.50588235, 0.5372549 ],
         [0.35686275, 0.43137255, 0.4627451 ]],

        [[0.25490196, 0.34117647, 0.36470588],
         [0.20784314, 0.29411765, 0.31764706],
         [0.28627451, 0.37254902, 0.39607843],
         ...,
         [0.37647059, 0.45098039, 0.48235294],
         [0.29411765, 0.36862745, 0.4       ],
         [0.2627451 , 0.3372549 , 0.36862745]]]


       [[[0.43921569, 0.40392157, 0.35294118],
         [0.44313725, 0.40784314, 0.35686275],
         [0.44705882, 0.41176471, 0.36078431],
         ...,
         [0.48235294, 0.44705882, 0.40784314],
         [0.49411765, 0.45882353, 0.41960784],
         [0.50196078, 0.46666667, 0.42745098]],
```

Fig 3. image digitization

Loss Calculation:

```
Code:
beta = 1.0
def rev_loss(true,pred):
loss = beta*K.sum(K.square(true-pred))
return loss

def full_loss(true,pred):
message_true, container_true = true[...,0:3], true[...,3:6]
message_pred, container_pred = pred[...,0:3], pred[...,3:6]
message_loss = rev_loss(message_true, message_pred)
container_loss = K.sum(K.square(container_true-container_pred))
loss = message_loss + container_loss
return loss
```

Fig 4: Pseudocode for the Loss Calculation

The text defines rev_loss and full_loss as two loss functions.

The rev_loss function performs a loss evaluation between real values and values that a model predicts. The customized loss measurement determines the total losses by counting squared value discrepancies between measured and predicted outcomes at a beta scaling level. The loss function presents utility for maximizing model performance at reversing steganography operations because it allows extraction of concealed messages from digital material containers.

A full_loss function determines the total loss that affects the steganography model. The input gets separated into message and container parts for both original and predicted data. Three input channels from the message stem form the message component and the following three channels construct the container component.

The Message Loss Calculation requires the rev_loss function to analyze both true and predicted message tensors. The function determines the loss that emerges from the message extraction functionality.

The calculation of container loss takes place through adding squared differences between actual and estimated values from both true and predicted components. The loss measures how effectively the model both protects the container structure and conceals the concealed information.

The Total Loss Calculation results from the combination of losses derived from container alteration measurements with losses measured from message alterations. Through this formulation the model learns to maximize both message hiding quality and container preservation.

The deep steganography model requires these loss functions to properly train its operation. The training process allows the model to acquire skills for message insertion into containers while minimizing their alteration and preserving their original content. The beta component of rev_loss enables specific project requirements to influence message extraction optimization by adjusting its priority in training.

B. Model Development

Image Steganography: Hide_Network

Sample Code:

```
def prep_and_hide_network(input_size):
  input_message = Input(shape=(input_size))
  input_cover = Input(shape=(input_size))
  x1 = Conv2D(50, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(input_message)
  x2 = Conv2D(10, (4,4), strides = (1,1), padding = 'same', activation = 'relu')(input_message)
  x3 = Conv2D(5, (5,5), strides = (1,1), padding = 'same', activation = 'relu')(input_message)
  x = concatenate([x1, x2, x3])

  x1 = Conv2D(50, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(x)
  x2 = Conv2D(10, (4,4), strides = (1,1), padding = 'same', activation = 'relu')(x)
  x3 = Conv2D(5, (5,5), strides = (1,1), padding = 'same', activation = 'relu')(x)
  x = concatenate([x1, x2, x3])
  x = concatenate([input_cover,x])
  image_container = Conv2D(3, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(x)
  encoder = Model(inputs = [input_message, input_cover],outputs = image_container)
  return encoder
```

Fig 5: Pseudocode for the Image Steganography

The prep_and_hide_network function implement a specialized neural network structure that specifically addresses image steganography because it specializes in hiding messages as "Hiding Images."

Inputs:

- The variable **input_message** works as the temporary storage point for messages which need to be hidden.
- A place named **input_cover** functions as the placeholder for inserting the image that will receive the hidden message.

Layers:

Multiple convolutional layers function successively to transform input message (input_message) through dedicated filter sets with different sizes (3x3, 4x4, 5x5). Filters in each layer operate at different spatial scales while employing specific filter dimensions (3x3, 4x4, 5x5). Diverse aspects from the input message can be extracted through parallel running convolutional layers (x1, x2, x3).

The outputs generated by concurrently operating convolutional layers become merged through concatenation (x) to unite respective features extracted by variable-sized filters.

The model obtains enhanced pattern detection abilities from higher-level representations by implementing convolutional layers as successive processing steps upon (x).

In the image_container output layer three convolutional filters produce the final results. The crucial stages of processing within the model fuse the concealment message with the cover image to form the steganographic image.

The Keras Model class enables the developer to create the encoder model with precise precision. The model operates with the input parameters input_message and input_cover to output the image_container result. The encoding process in steganography gets realized through this model which demonstrates the occult process of inserting message content into cover images to generate steganographic creations.

The encoder model function provides a return statement that delivers the neural network architecture designed especially for the covert processing of messages into cover images.

This function presents a neural network model that achieves peak performance in image steganography applications. The neural network uses convolutional layers to process input messages for hiding them in cover images which results in steganographic compositions while serving as a valuable tool for visual message obscurity.

Image Steganography: Reveal Network

Sample Code:

```
def reveal_network(input_size, fixed=False):
  reveal_input = Input(shape=(input_size))
  input_with_noise = GaussianNoise(0.01)(reveal_input)

  x1 = Conv2D(50, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(input_with_noise)
  x2 = Conv2D(10, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(input_with_noise)
  x3 = Conv2D(5, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(input_with_noise)
  x = concatenate([x1, x2, x3])

  x1 = Conv2D(50, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(x)
  x2 = Conv2D(10, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(x)
  x3 = Conv2D(5, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(x)
  x = concatenate([x1, x2, x3])
  message = Conv2D(3, (3,3), strides = (1,1), padding = 'same', activation = 'relu')(x)
  reveal =  Model(inputs = reveal_input,outputs = message)
  return reveal
```

Fig 6: Pseudocode of Reveal Network

The **reveal_network** function builds an advanced neural network architecture that detects hide messages in steganographic images. An examination of the structure along with features of this function shows:

Inputs:

- **reveal_input**: The hidden embedding area functions as an empty section that holds the concealed message of the steganographic image.

Noise Injection: GaussianNoise(0.01) represents an essential network layer which adds Gaussian noise with standard deviation set to 0.01 in the steganographic image. By implementing this strategic step the model demonstrates enhanced resistance against input image fluctuations which should strengthen its ability to generalize.

Layers:

Convolutional Layers: Identical to the previous network design the architecture utilizes multiple convolutional layers to analyze the steganographic image called reveal_input. The network extracts vital features through its multiple layers that facilitate_hidden_message discovery. Distinct parallel convolutional layers (x1, x2, x3) use multiple 3x3 filters for spatial scale manipulation during the extraction process.

Concatenation: The outputs from parallel convolutional layers get harmoniously combined (x) to merge features extracted with different filter dimensions.

Intermediate Processing: An additional set of convolutional layers identifies advanced representations and patterns inside the steganographic image after combining features (x).

Output Layer: A convolutional layer at the conclusion represents the message with three built-in filters. The uppermost layer exposes the extracted message which emerges from the steganographic image.

Model Definition: The reveal network derives its definition from the Keras Model class. The reveal_input independent variable serves as input while the program generates message as its primary output. The model reflects the sophisticated steganography decoding mechanism which recovers the hidden message from the stego images.

Fixed Noise (Optional): During training the fixed parameter operates as a boolean flag to control the steady state of injected noise. By setting it True the injected noise holds its value permanently thus possibly leading to regularization benefits.

The function creates an exact neural network design to detect hidden data concealed in steganographic images. The convolutional layers enable the approach to analyzesteganographic images effectively while extracting hidden content and demonstrating how steganographic decoding works. The neural model achieves high accuracy with robustness by including Gaussian noise along with parallel convolutional layers during its operation to decode steganographic messages.

Image Steganography: Model Compilation

Sample Code:

```
shape = input_S.shape[1:]
input_message = Input(shape = (shape))
input_container = Input(shape = (shape))
prep_and_hide = prep_and_hide_network(shape)
reveal = reveal_network(shape)
reveal.compile(optimizer = 'adam',loss = rev_loss)
reveal.trainable = False
output_container = prep_and_hide([input_message, input_container])
output_message = reveal(output_container)
deep_stean = Model(inputs = [input_message, input_container],
                   outputs = concatenate([output_message, output_container]))
deep_stean.compile(optimizer = 'adam',loss = full_loss)
```

Fig 5: Pseudocode for the Model Compilation

This code element merges two previously created networks (prep_and_hide_network and reveal_network) into a single deep steganography model structure which works for training along with inference operations. A detailed explanation follows regarding these procedures:

Input Definitions:

- The shape = input_S.shape[1:] operation retrieves input_S dimension characteristics to show the images involved in stegano-operations.
- The system accepts input_message and input_container through two placeholders whose dimension characteristics are set by the shape parameter.

Network Instantiation:

- prep_and_hide = prep_and_hide_network(shape): Instantiates the encoder network (prep_and_hide_network) with the specified input shape.

- reveal = reveal_network(shape): Similarly, instantiates the decoder network (reveal_network) with the same input shape.

Compilation:

- The compilation of decoder network (reveal) utilizes Adam optimizer with rev_loss function. The decoder becomes ready for training after defining its optimization algorithm and selected loss minimization function through this configuration.

Freezing the Decoder:

- reveal.trainable = False: Freezes the weights of the decoder network (reveal). Training operation for the decoder network is prevented by setting trainable to False which maintains its static state throughout the entire deep steganography model.

Model Composition:

- The prep_and_hide function calculates the output container image through its process of input_message and input_container with the encoder network.
- The output message emerges from reveal(output_container) as this function uses output container image data to run it through the decoder network.

Model Definition:

- The deep_stegan model vanishes through the Keras Model class as the unified deep steganography model that processes input_message and input_container while generating output_message and output_container as outputs. The specification of this definition states the input combination between message and container images and establishes output elements which include the recovered message and altered container images. The final output emerges from concatenating the output_message designed by the decoder with output_container created by the encoder through the channel axis.

Compilation of the Deep Steganography Model:

- The deep steganography model (deep_stegan) receives Adam optimizer and full_loss function when compiled here. The model obtains training capabilities through this setup which specifies both the optimization method and the loss criterion that should reach minimum levels during the training period.

This segment of code creates the deep steganography model through a structured combination of encoder and decoder networks for both training and inference operations. The model goes through training preparation by utilizing appropriate optimizers together with loss functions to achieve training readiness.

C. Model Summary

```
 model (Functional)          (None, 64, 64, 3)            293273    ['input_1[0][0]',
                                                                     'input_2[0][0]']

 model_1 (Functional)        (None, 64, 64, 3)            155938    ['model[0][0]']

 concatenate_13 (Concatenat  (None, 64, 64, 6)            0         ['model_1[0][0]',
 e)                                                                  'model[0][0]']

==================================================================================
Total params: 449211 (1.71 MB)
Trainable params: 293273 (1.12 MB)
Non-trainable params: 155938 (609.13 KB)
```

Users can obtain structural information along with parameter counts through the implementation of the model.summary() command. The deep learning model called model includes 293273 trainable parameters whereas model_1 consists of 155938 parameters which cannot be trained. The count of trainable parameters between these models demonstrates that model_1 integrates pre-trained sections and frozen weights because of the fixed parameter in the reveal_network function.

D. Model Training:

Code:

```
for i in range(100):
    batch_message = input_S[i*batch_size:min((i+1)*batch_size, m)]
    batch_cover = input_C[i*batch_size:min((i+1)*batch_size, m)]
    container = prep_and_hide.predict([batch_message, batch_cover])
    f_loss = deep_stean.train_on_batch(x=[batch_message, batch_cover],
                                       y=np.concatenate((batch_message, batch_cover), axis=3)
    r_loss = reveal.train_on_batch(x=container,
                                   y=batch_message)
    f_loss_mean += f_loss
    r_loss_mean += r_loss
# Calculate mean losses for the epoch
f_loss_mean /= itera
r_loss_mean /= itera
# Append mean loss to history for plotting
full_loss_history.append(f_loss_mean)
reveal_loss_history.append(r_loss_mean)

# Adjust learning rate based on the schedule
K.set_value(deep_stean.optimizer.lr, lr_schedule(epoch))
K.set_value(reveal.optimizer.lr, lr_schedule(epoch))
print(f'Epoch = {epoch} | Mean full loss = {f_loss_mean} | Mean reveal loss = {r_loss_mean}')
```

Fig 6: Pseudocode for the Model Training

Through this code block the deep steganography model training loop functions while the lr_schedule mechanism controls the learning rate adjustments throughout training. Each detailed aspect of the procedures follows in this paragraph.

Learning Rate Schedule Function (lr_schedule):

- The function operates on an epoch index to establish a learning rate through defined schedules.
- The function first utilizes a higher learning rate value of 0.001 during the first 200 epochs when epoch_idx remains below 200.
- The code reduces the learning rate from 0.001 to 0.0003 throughout epoch ranges 201 to 400.
- The learning rate decreases to 0.0001 starting from epoch_idx 200 to epoch_idx 400.
- A learning rate set to 0.00003 applies starting from epoch 600 through the end of training time.

Initialization:

- The program defines m as an initializing parameter that represents training set sample count (input_S).
- A loss_history list is established to store loss values from training.
- The program defines batch_size as 32 to achieve the required size of training batches.

Training Loop (for epoch in range(1000):):

- The loop extends its operation throughout 1000 epochs for training model functions.
- Within each epoch:
    - Randomness is added through shuffling of the training data consisting of input_S and input_C.
    - The calculated number of iterations (itera) depends on what batch size has been selected.

- Each batch accumulation step results in calculated average values for f_loss_mean and r_loss_mean of the losses f_loss and r_loss.
- For each batch:
    - The Message and cover images that will be used should be chosen specifically for the chosen batch.
    - The batch transforms into a container image when it flows through the encoder operating as prep_and_hide.predict.
    - The entire steganography model (deep_stegan) learns on each batch by reducing prediction errors of both message and container images.
    - During training the decoder receives the container image input to reduce the difference between recovered message and its intended value.
- The learning rates for steganography model and decoder components receive updates through their respective learning rate schedules (lr_schedule(epoch)).

Printing and Logging:

- The training process is monitored through intelligent printing of loss values that occur for individual batches and epochs.
- A complete overview emerges through the presentation of epoch mean losses to the user.
- Each epoch includes learning rate updates which are visible in the printout.

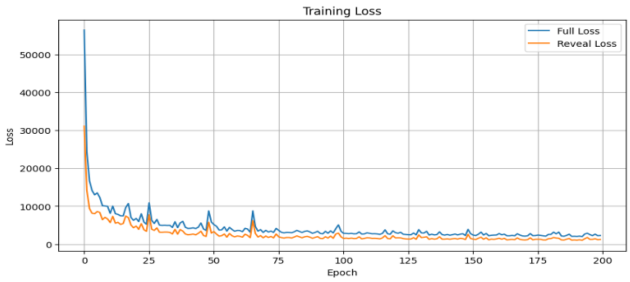


Fig 7. Model Performance

The deep steganography model receives precise training through this code block together with a dynamic learning rate mechanism to optimize its learning process throughout the training period. The system performs multiple batches of epochs while advancing the model parameters by minimizing the targeted loss function through each cycle.

The depicted loss versus epochs relationship shows that training loss reduces continuously during the first 200 epochs thus providing important feedback about model training operations. The model shows increasing predictive accuracy through successive epochs which leads to its downward trend in this loss pattern.

The preliminary stages start with high loss because the random weight initialization produces predictions that stand far apart from actual measurements. The optimization algorithms SGD and Adam adjust the model weights through training which results in decreased loss due to the minimization of value differences between predictions and actual results.

The model shows effective learning abilities because loss consistently decreases which indicates the successful identification of patterns within training data. Systematic improvements of parameters throughout 200 epochs show how the model learns and evolves its parameters effectively which showcases its efficient training process. The examination of loss during training sessions functions as an essential measuring tool that helps assess when models achieve convergence and improve performance.

E. Evaluation

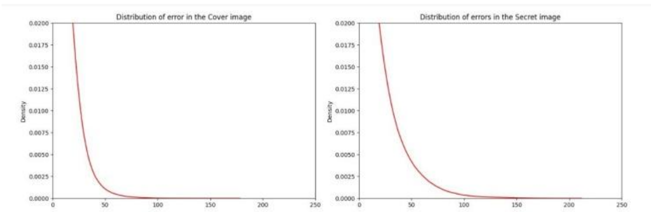
Distribution of Errors:

Code:

```
def pixel_errors(input_S, input_C, decoded_S, decoded_C):
    see_Spixel = np.sqrt(np.mean(np.square(255*(input_S - decoded_S))))
    see_Cpixel = np.sqrt(np.mean(np.square(255*(input_C - decoded_C))))
    return see_Spixel, see_Cpixel

def pixel_histogram(diff_S, diff_C):
    diff_Sflat = diff_S.flatten()
    diff_Cflat = diff_C.flatten()
    fig = plt.figure(figsize=(15, 5))
    a=fig.add_subplot(1,2,1)
    imgplot = plt.hist(255* diff_Cflat, 100, alpha=0.75, facecolor='red')
    a.set_title('Distribution of error in the Cover image.')
    plt.axis([0, 250, 0, 0.2])
    a=fig.add_subplot(1,2,2)
    imgplot = plt.hist(255* diff_Sflat, 100, alpha=0.75, facecolor='red')
    a.set_title('Distribution of errors in the Secret image.')
    plt.axis([0, 250, 0, 0.2])
    plt.show()
```

KDE enhances error distribution visualization to produce better clarity along with information about error patterns caused by embedding and extraction operations. Improved visualization through KDE enables stakeholders to both detect errors as well as find distribution patterns and effectively share research conclusions..



Peaks and fluctuations appear in the error distribution of the cover image because the embedding process creates significant modifications to original content. The peaks which appear in the error distribution correspond to regions with intense high-frequency or high-intensity characteristics especially edges and texture patterns where secret data embedding produces notable deviations from the original pixel values.

The error distribution in the secret image reflects the differences that appear between initial hidden data and recovered data during steganographic operations. A smooth error distribution estimation through KDE overlay reveals hidden error patterns at different intensity levels thus helping to detect possible weak spots in concealed information.

International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Code:

```
n = 6
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
def show_image(img, n_rows, n_col, idx, gray=False, first_row=False, title=None):
    ax = plt.subplot(n_rows, n_col, idx)
    if gray:
        plt.imshow(rgb2gray(img), cmap = plt.get_cmap('gray'))
    else:
        plt.imshow(img)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    if first_row:
        plt.title(title)
plt.figure(figsize=(14, 15))
rand_indx = [random.randint(0, 1000) for x in range(n)]
for i, idx in enumerate(rand_indx):
    n_col = 6 if SHOW_DIFF else 4
    show_image(input_C[idx], n, n_col, i * n_col + 1, gray=SHOW_GRAY, first_row=i==0, title='Cover')
    show_image(input_S[idx], n, n_col, i * n_col + 2, gray=SHOW_GRAY, first_row=i==0, title='Secret')
    show_image(decoded_C[idx], n, n_col, i * n_col + 3, gray=SHOW_GRAY, first_row=i==0, title='Encoded Cover')
    show_image(decoded_S[idx], n, n_col, i * n_col + 4, gray=SHOW_GRAY, first_row=i==0, title='Decoded Secret')
plt.show()
```

The code segment displays original cover and secret images while showing their decoded counterparts for visual performance evaluation of the deep steganography model.

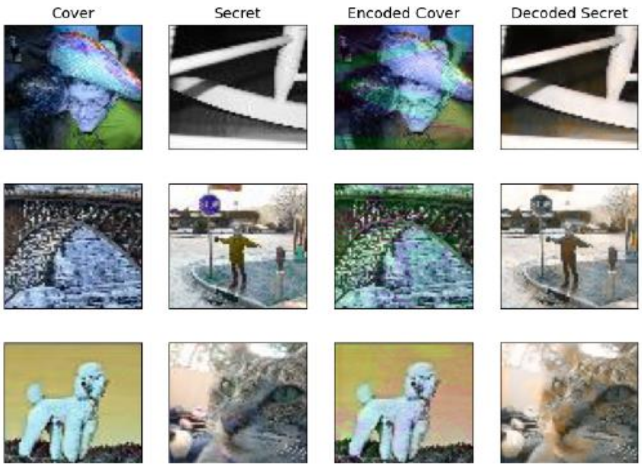


Fig 8. Effectiveness and fidelity of the deep steganography

The show_image function shows images in grid format using Matplotlib. A subset of randomly chosen samples appears for assessment purposes. The program shows original cover and secret images first followed by their decoded versions for each selected index. The difference images between original and decoded versions appear when such comparison is requested by the user. The visual assessment tool gives a quality perspective on how well the model conceals and uncovers secret information from within covered images to help determine deep steganography's effectiveness and faithfulness.

VII. CONCLUSIONS

This research illustrates the potential of deep steganography in securely hiding information within digital images using neural networks. Utilizing the Tiny ImageNet dataset, the model effectively embedded and retrieved hidden data while minimizing message loss and visual distortion. The preprocessing phase normalized pixel values, enabling seamless integration with neural networks. Structured arrays were used to manage cover images (`input_C`) and secret messages (`input_S`), allowing the model to generalize across diverse image classes.

The architecture consists of two key networks: the **Hide Network** (encoder) and the **Reveal Network** (decoder). The encoder embeds secret messages using convolutional layers with varying filter sizes to maintain the appearance of the original image. The decoder applies the same convolutional structure to extract hidden content with precision. Two tailored loss functions—`rev_loss` and `full_loss`—were employed to optimize the learning process. While `rev_loss` focused on accurate message retrieval, `full_loss` ensured both message recovery and container integrity.

A dynamic learning rate schedule contributed to smoother training and convergence by adjusting learning rates over time. The resulting model demonstrates strong encryption performance and message fidelity, proving that deep learning can significantly enhance steganographic security. This work offers a foundational framework for building advanced, undetectable image-hiding systems in modern communication environments.

REFERENCES

[1] Ntivuguruzwa, J. D. L. C., & Ahmad, T. (2023). A CNN architecture for improving spatial steganography detection.

[2] Kumar, M., Rao, S. P., & Choudhary, R. (2020). CNN-based approaches for image steganography. Sensors.

[3] Hashemi, F., Majidi, A., &Khorashadizadeh, R. (2022). Color image steganography using autoencoders and ResNet.

[4] Laxmi, R., & Rajkumar, D. (2023). A review of classical and DL-based steganography methods.

[5] Kaneria, S., &Jotwani, V. (2024). Comparative analysis of deep learning encoders for image steganography.

[6] Płachta, M., Rudziński, T., & Śmigielski, D. (2022). Detecting steganographic content in JPEG images using ensemble classifiers.

[7] Yola, A. et al. (2023). Hybrid activation functions in CNNs for steganalysis.

[8] Bhatt, A., Patel, K., & Shah, J. (2024). Deep steganography using CNNs and machine learning.

[9] Hegarty, C., & Keane, M. (2020). CNNs for detecting covert data in images.

[10] Xie, G., Ren, J., et al. (2019). Comparing traditional and DL-based methods for image steganalysis.