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

Mr.Ved Rajdeep[1], Asst. Prof. Nimesh Vaidya[2], Dr. Vijaykumar B Gadhavi[3]

[1]PG Scholar, [2]Assistant Professor & HOD, [3]Associate Professor & Dean
[1, 2, 3]Faculty of Engineering, Computer Engineering Department Swaminarayan University, India

Abstract

This paper explores advanced steganographic techniques that leverage digital images as carriers to conceal secret information within visually benign files. The core motivation lies in steganography's inherent capability to ensure confidentiality through invisibility, offering an innovative and secure approach to safeguarding sensitive data from potential cyber threats. To this end, a sophisticated deep steganography framework is proposed, combining machine learning with steganographic principles to enable the seamless embedding of one image within another. The design ensures that the concealed image remains indistinguishable to the human eye while preserving the overall visual integrity of the cover image.

The primary objective of this research is to enhance the effectiveness of data embedding while maintaining high visual quality, addressing critical challenges in the domain of information security. Through the integration of adaptive machine learning algorithms, the model is capable of dynamically adjusting the payload size in accordance with the structural characteristics of the cover image. This adaptability reinforces the system's robustness, making it more resilient against unauthorized detection and extraction.

The proposed model employs Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) in a joint training strategy, guided by customized loss functions designed to minimize distortion and maximize embedding fidelity. The system's performance is evaluated using objective metrics such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and message recovery rate, supplemented by visual inspection and comprehensive robustness testing. Additional assessments of computational efficiency and resistance to steganalysis further validate the model's reliability and applicability.

Ultimately, the deep steganography system introduced in this study demonstrates the potential of modern neural network architectures in constructing secure, covert communication channels. Its development and empirical validation illustrate its practical utility in preserving data privacy and integrity across increasingly interconnected digital ecosystems.

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

1. Introduction

Steganography, the art of concealing messages within innocuous media, traces its origins back to around 440 BCE. Early practices involved physical methods such as writing on wax tablets and utilizing invisible inks to transmit hidden information. Over the centuries, this technique has evolved alongside technological advancements, culminating in sophisticated digital applications aimed at protecting information against unauthorized access and emerging cybersecurity threats.

The term "steganography" is derived from the Greek words *steganos* (meaning "covered") and *graphie* (meaning "writing"), encapsulating the concept of hidden communication. In modern times, steganography plays a critical role in digital security, employing advanced algorithms to shield sensitive data from detection and hacking attempts.

Among various digital steganographic methods, image steganography has emerged as the most prevalent due to images' high storage capacity and widespread availability across digital platforms. This technique involves subtle modifications to pixel values of a host image to embed secret information. Predominantly, two categories define embedding strategies: spatial domain and transform domain techniques. In the spatial domain, methods such as Least Significant Bit (LSB) manipulation are employed, while in the transform domain, Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) techniques leverage frequency-based modifications to enhance resilience against common image processing operations like compression and scaling.

The integration of deep learning has markedly transformed the landscape of steganography research in recent years. Neural network architectures, particularly Convolutional Neural Networks (CNNs), have proven highly effective in learning complex pixel-level features essential for robust data embedding and extraction. These deep learning models offer adaptive frameworks capable of surpassing traditional methods in terms of imperceptibility, resilience, and detection avoidance.

Continuous advancements in neural network technologies have not only improved the visual quality of stego-images but also enhanced the security of embedded information, ensuring that concealed data remains indistinguishable to human observers and analytical tools.

This research endeavors to develop an advanced deep steganography model that addresses existing technical limitations by leveraging the capabilities of deep learning. Future work in this domain aims to build upon these foundations, striving for greater accuracy, robustness, and security in the face of evolving cyber threats, thereby supporting the ongoing development of secure digital communication frameworks.

2. Objective

The principal aim of this research is to design and implement a robust deep steganography framework that leverages neural networks to enhance image-based data concealment and retrieval. The proposed system strives to achieve a high degree of security, imperceptibility, and adaptability—three essential pillars for modern steganographic applications in dynamic digital environments. This study addresses current technological challenges by constructing a deep neural network architecture that balances detectability prevention with system resilience and operational flexibility.

At the core of the proposed methodology lies a dual-network architecture:

- **Hiding Network (Encoder):** This network is responsible for embedding the secret image into the cover image while preserving the visual integrity of the host.
- **Revealing Network (Decoder):** Designed to accurately extract the embedded secret image, ensuring that the retrieved data closely matches its original form without degradation.

The goal is to achieve an optimal balance between embedding strength and image quality, creating stego-images that remain indistinguishable to both human observers and sophisticated steganalysis tools. To this end, novel loss functions are introduced during the training phase to minimize distortion and ensure that both the container and the hidden image retain high visual fidelity.

A key innovation in this framework is the incorporation of **dynamic payload adaptation**, wherein the system intelligently adjusts the amount of hidden data based on the content complexity and spatial characteristics of the cover image. This ensures compatibility with a wide range of image resolutions, formats, and content types, thereby broadening the system's applicability.

Moreover, the proposed approach emphasizes **robustness against steganalytic attacks and image processing alterations**. The model is engineered to resist common transformations such as compression, scaling, and noise injection, while also maintaining resilience to statistical detection techniques.

Ultimately, this research aims to establish a foundational framework for adaptive, secure, and low-detectability steganography. The developed system has potential applications in secure communication, data confidentiality, digital watermarking, and intellectual property protection, contributing to the advancement of steganographic technologies in real-world scenarios.

3. Problem Statement

3.1 Limitations of Traditional Steganographic Techniques

Despite their foundational role in data concealment, conventional steganographic methods exhibit critical shortcomings in addressing modern digital security challenges. Most notably, these techniques suffer from limited data embedding capacity and lack the robustness required to protect against sophisticated digital forensics and adversarial attacks. The restricted payload size severely constrains their applicability in scenarios requiring large-scale or high-security data hiding. Furthermore, the growing efficacy of forensic tools and heuristic-based detection mechanisms significantly reduces the effectiveness of classical steganography, rendering it insufficient for current security demands. These limitations underscore the need for a paradigm shift toward more adaptive and resilient approaches.

3.2 The Need for Advanced Steganographic Solutions

As cybersecurity threats grow more complex and pervasive, there is an urgent demand for advanced steganographic methods capable of withstanding modern attack vectors. Traditional approaches fail to offer adequate protection against adaptive adversaries and advanced detection techniques. In response, the integration of deep learning frameworks has emerged as a promising direction. These models leverage the predictive and adaptive capabilities of neural networks to enhance both the concealment and retrieval processes, resulting in systems with superior resilience, security, and operational efficiency. Deep learning-driven steganography offers a pathway to developing intelligent, scalable, and attack-

tolerant solutions.

3.3 Objectives for Deep Learning-Based Steganographic Models

This study aims to design and implement a novel steganographic framework built upon deep learning infrastructures, primarily using Convolutional Neural Networks (CNNs). The proposed model seeks to maximize data hiding efficiency while minimizing visible distortion in the cover image. By overcoming the limitations of traditional methods, the framework aspires to enhance stealth, increase payload capacity, and provide robust protection against detection and extraction attempts. These improvements are especially critical in applications where data security and integrity are paramount, such as confidential communications and digital watermarking.

3.4 Challenges in Algorithm Optimization and Robustness

One of the primary challenges in developing an effective steganographic model lies in optimizing the embedding algorithm to maintain both imperceptibility and robustness. Embedding large volumes of hidden data without compromising the structural fidelity of the cover image is inherently complex. Achieving this balance demands fine-tuning loss functions, managing adversarial noise, and reinforcing the system against typical image distortions such as compression, resizing, and added noise. The dual objective of high-security assurance and minimal perceptual degradation presents a significant technical hurdle that must be addressed to ensure practical deployment.

3.5 Towards a Practical and Accessible Steganographic System

Beyond theoretical advancements, the ultimate goal of this research is to develop a user-friendly, deployable steganographic tool that can be used by both technical and non-technical stakeholders. The proposed solution emphasizes simplicity in interface design while preserving the sophisticated underlying mechanisms required for secure data hiding. Such an approach ensures accessibility for a wide range of users—individuals, enterprises, and security agencies—enabling real-world application of deep learning-based steganography in various domains including secure communications, digital rights management, and intellectual property protection.

4. Literature Review

Recent advancements in the fields of steganography and steganalysis have been significantly influenced by the integration of deep learning, particularly Convolutional Neural Networks (CNNs). These techniques have been employed to overcome the inherent limitations of traditional approaches, offering enhanced accuracy, adaptability, and robustness.

Ntivuguruzwa et al. (2024) demonstrated the effectiveness of CNN architectures in steganalysis, showing that these models can autonomously learn spatial features and facilitate the detection of concealed data within stego-images. By combining feature extraction and classification within a unified deep learning pipeline, their approach significantly improved detection accuracy. Similarly, Kumar et al. (2020) emphasized the advantages of CNNs over conventional classifiers, particularly due to their capacity for self-learning, which enables reliable performance even at low embedding rates.

Further innovations include the use of convolutional autoencoders and ResNet models for steganographic tasks, as proposed by Hashemi et al. (2022). Their framework achieved exceptional

imperceptibility with a Peak Signal-to-Noise Ratio (PSNR) exceeding 40 dB and a Structural Similarity Index Measure (SSIM) above 0.98, indicating high visual quality and strong resilience against steganalysis techniques. In another study, Kumar et al. (2020) introduced a dual-network configuration comprising H-net and R-net, which enabled effective embedding and retrieval of hidden data while maintaining low visual distortion.

Additional enhancements in deep steganography systems have involved hybrid activation functions, adaptive payload mechanisms, and customized loss functions specifically tailored to preserve image quality during embedding. These improvements collectively contributed to increased performance across various benchmarks.

Many of these approaches have been rigorously tested on well-established datasets such as BOSSBase, COCO, and CelebA, demonstrating their generalizability and robustness across diverse image types and conditions.

In summary, the convergence of deep learning and steganography has led to the development of secure, adaptive, and high-capacity frameworks suitable for real-world applications, particularly in scenarios requiring discreet and reliable communication of sensitive information

Table 1. 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, Majidi, & Khorashadizadeh (2022)	Improved color image steganography using autoencoders and ResNet	Convolutional autoencoders for feature extraction and ResNet for embedding/extraction	High imperceptibility (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	Xie, Ren, et al. (2019)	Comparison of traditional and DL-based steganalysis methods	Conventional methods (SPAM, SRM) vs. DL (Ye-Net, SRNet)	Highlighted DL's superior performance in feature extraction and classification; emphasized combining traditional insights with DL for improved accuracy.

5. Methodology

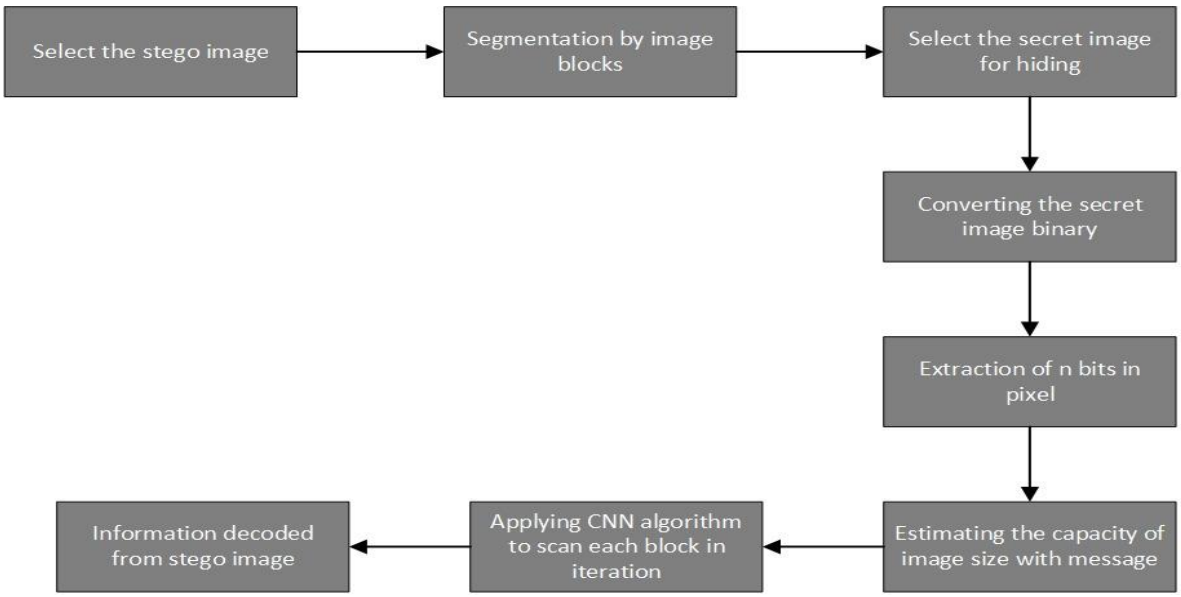


Fig. 1 Proposed Flow of the Model

The methodology adopted in this study involves the design, training, evaluation, and optimization of a deep learning-based steganography model, with a focus on practical applicability, robustness, and accuracy. The complete pipeline is outlined as follows:

1. Data Collection and Preprocessing

A foundational step in building an effective steganographic model is the preparation of a diverse and representative dataset. The dataset comprises pairs of **cover images** and **secret images**, each selected to ensure a variety of image types, resolutions, and content. To enable uniformity and compatibility during training, the following preprocessing techniques are applied:

- **Resizing:** All images are resized to a consistent resolution to standardize input dimensions across the model.
- **Normalization:** Pixel values are scaled (e.g., to the range [0,1]) to facilitate smoother and faster convergence during neural network training.

This dual preprocessing approach ensures that both cover and secret images are in optimal condition for model learning and generalization.

2. Model Architecture Selection

Two deep learning architectures form the basis of the proposed system:

- **Convolutional Neural Networks (CNNs):** Chosen for their proven ability to extract and analyze spatial features, CNNs serve as the primary architecture for both embedding and extraction.
- **Recurrent Neural Networks (RNNs):** Incorporated for their ability to capture sequential patterns, particularly beneficial in managing dependencies and learning temporal variations during the data hiding and recovery process.

This dual-framework design supports accurate learning of complex image features and structural patterns.

3. Training Strategy

The training process involves iterative fine-tuning of model parameters to enhance both the embedding and extraction performance. A composite loss function is utilized to balance the dual objectives of:

- **Minimizing visual distortion** in the cover image after embedding.
- **Maximizing recovery accuracy** of the hidden secret image.

Training is conducted across multiple epochs with optimization algorithms (e.g., Adam), enabling the generation of stego-images that are perceptually indistinguishable from their original counterparts.

4. Embedding Algorithm Optimization

To enhance embedding efficiency and reduce perceptual artifacts, the embedding algorithm undergoes rigorous optimization. This involves:

- **Regularization techniques** to prevent overfitting.
- **Hyperparameter tuning** to adjust key settings like learning rate, batch size, and filter depth.
- **Architectural refinements** aimed at improving feature representation and minimizing structural loss in the output images.

These steps collectively improve the fidelity and security of the steganographic process.

5. **Robustness Evaluation**

Robustness testing is an essential component of model validation. The system is evaluated under various adversarial scenarios, including:

- **Image processing operations** such as resizing, compression, and noise addition.
- **Steganalysis attacks** using statistical and heuristic-based detection models.

Performance metrics are gathered to assess the model's ability to preserve hidden data across environmental fluctuations and digital transformations.

6. **Development of a User-Friendly Interface**

A practical software tool has been developed to demonstrate the model's usability in real-world settings. Key features include:

- **Graphical User Interface (GUI):** Simplifies the embedding and extraction processes, allowing users with minimal technical expertise to utilize the system.
- **Usability Testing:** Comprehensive evaluation by both technical and non-technical users to confirm accessibility and operational reliability.

This ensures that the system is not only technically sound but also intuitively usable.

7. **Performance Evaluation and Benchmarking**

Model performance is assessed using both quantitative and qualitative measures. Key evaluation criteria include:

- **PSNR and SSIM** for image quality analysis.
- **Message Recovery Rate** to measure extraction accuracy.
- **Comparative Benchmarking** with existing methods to highlight improvements in efficiency, security, and imperceptibility.

This multi-faceted evaluation confirms the effectiveness of the proposed solution.

8. **Documentation and Reproducibility**

Comprehensive documentation is maintained throughout the development cycle, including:

- **Experimental design and parameters**
- **Training logs and model configurations**
- **Source code and scripts for model reproduction**

Final reporting includes an in-depth summary of results, a critical discussion of findings, and a roadmap for potential enhancements, thereby promoting reproducibility and future extensions of the research.

6. 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.

6.1. 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 fulfill steganographical 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.

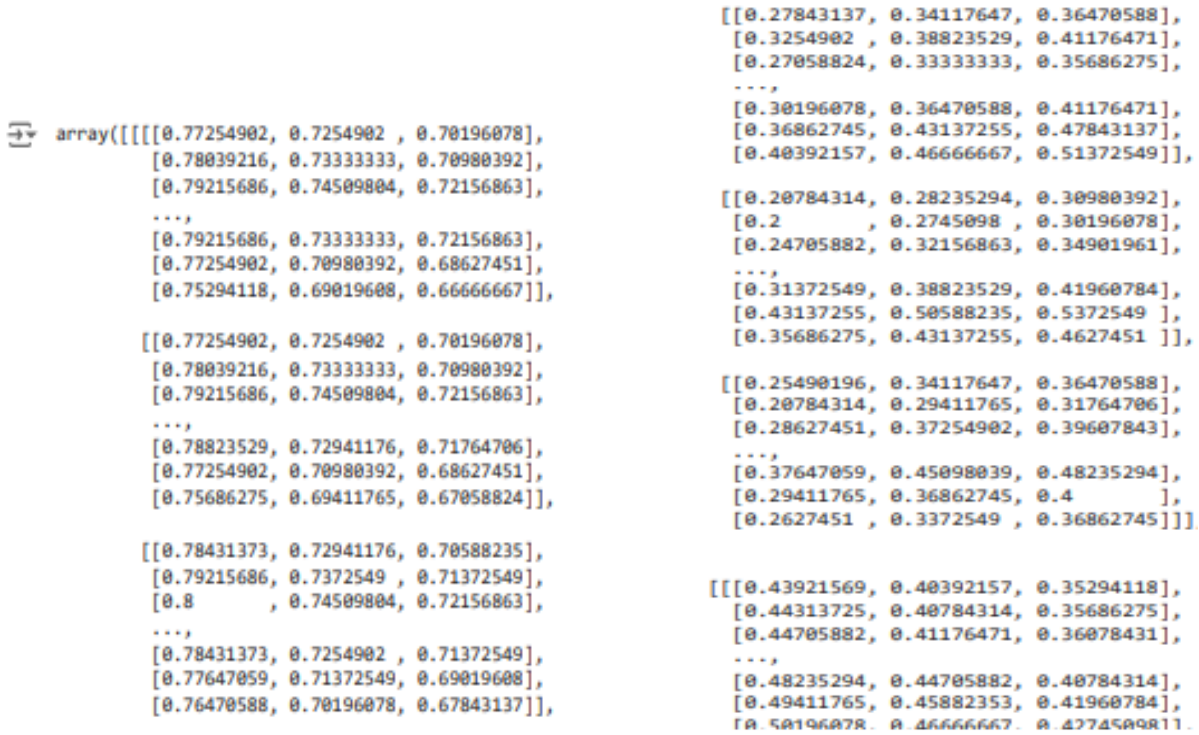


Fig 3. image digitization

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.

6.2 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 analyze steganographic 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 7: 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.

6.3 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.

6.4 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 8: 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.

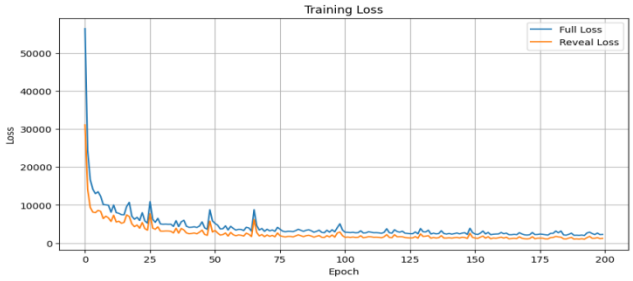


Fig 9. Model Performance

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.

6.5 Evaluation

Distribution of Errors:

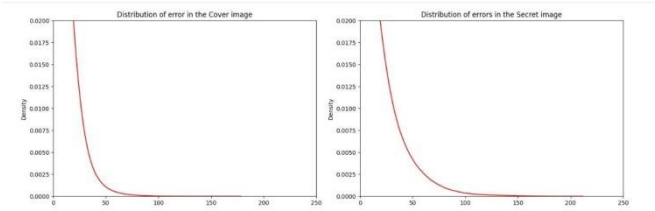
Code:

```
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.

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 10. 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.

7.Conclusion

This study demonstrates the effectiveness of deep learning techniques—particularly neural networks—in enabling secure information embedding within digital images. By leveraging the Tiny ImageNet dataset, the proposed model successfully achieved high-fidelity image steganography, ensuring minimal perceptual distortion and message loss during both embedding and extraction phases. The preprocessing pipeline included normalization of pixel values, allowing for smoother integration with neural network architectures and promoting generalizability across a variety of image types.

The model architecture was composed of two primary neural components: the **Hiding Network** (encoder) and the **Revealing Network** (decoder). The encoder utilized convolutional layers with varying kernel sizes to discretely insert secret messages into the cover images without compromising visual quality. In contrast, the decoder applied a similar layered convolutional structure to recover the concealed content with high accuracy. Custom loss functions—rev_loss for message recovery and full_loss for balancing recovery with image integrity—were strategically employed to guide the learning process.

To enhance training efficiency and model convergence, a dynamic learning rate scheduling mechanism was implemented. This adjustment strategy allowed the model to adapt throughout the training epochs, leading to more stable optimization and improved performance.

Overall, the model achieved robust steganographic performance, balancing imperceptibility with message fidelity. These results underscore the potential of deep learning to advance the field of information hiding, laying the groundwork for future systems capable of embedding confidential data invisibly and securely in digital media. The proposed framework provides a viable pathway for the development of practical, high-security steganographic tools applicable to modern communication infrastructures.

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.