Application of Singular Value Decomposition for Image Steganography in Regions of Interest with YOLOv8 Object Detection

Refki Alfarizi - 13523002[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]refkialfarizi46@gmail.com, 13523002@std.stei.ac.id

***Abstract*—Image steganography, the practice of embedding concealed information within digital images, is essential for secure communication and data protection. This paper introduces an approach that combines Singular Value Decomposition (SVD), a fundamental linear algebra technique, with object detection algorithms to enhance steganographic methods. By utilizing YOLOv8 for accurate detection of Regions of Interest (ROIs) within an image, the method restricts data embedding to these specific areas, potentially improving both the capacity and imperceptibility of hidden information. The application of SVD allows robust data embedding by decomposing image matrices into singular vectors and values, facilitating the manipulation of image components with minimal distortion. Experimental results indicate that our SVD-based steganography within detected ROIs demonstrates promising performance in data embedding capacity and resilience against common image processing attacks. This work highlights the integration of linear algebra and computer vision techniques, offering a framework for secure and efficient image steganography.**

***Keywords*—Image Steganography, Region of Interest, Singular Value Decomposition, YOLOv8.**

I. INTRODUCTION

In the digital era, the secure transmission of information is paramount, driving the need for advanced techniques in data protection. Image steganography, the practice of embedding concealed information within digital images, offers a covert method for secure communication by hiding data in a manner that is imperceptible to the human eye. Unlike encryption, which merely obscures the content of the data, steganography ensures the undetectability of the data's existence, thereby providing an additional layer of security. However, the effectiveness of steganographic methods hinges on their ability to embed data without introducing noticeable distortions to the host image, a challenge that necessitates sophisticated techniques for optimal performance.

Singular Value Decomposition (SVD), a fundamental concept in linear algebra, has emerged as a powerful tool in image processing applications, including steganography. By decomposing an image matrix into its singular vectors and singular values, SVD facilitates the manipulation of specific image components with minimal impact on overall image quality. This property makes SVD particularly suitable for data embedding, as it allows for the concealment of information within the less perceptually significant singular values. Concurrently, advancements in computer vision, particularly in object detection algorithms like YOLOv8, have enabled precise identification of Regions of Interest (ROIs) within images. Integrating object detection with SVD-based steganography presents a synergistic approach, where data embedding is confined to salient regions, thereby optimizing both data capacity and imperceptibility.

II. THEORETICAL BASIS

A. Matrix

A matrix is a rectangular array of numbers. The numbers in the array are called the entries of the matrix [1]. Denoted typically by uppercase letters (e.g., A, B, C), matrices are pivotal in modeling and solving linear equations and represent various data structures in computer science. Formally, an $m \times n$ matrix A is defined as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad (1)$$

where a_{ij} represents the element in the i-th row and j-th column of A. Matrices facilitate the representation and manipulation of data in multidimensional spaces, making them indispensable for tasks such as transformations, system modeling, and data analysis.

B. Matrix Representation of Images

Digital images are represented in rows and columns [2] which inherently represented as matrices, where each image is structured as a two-dimensional matrix of pixel intensity values. For grayscale images, each element p_{ij} of the matrix corresponds to the intensity of the pixel located at the i-th row and j-th column. In the case of color images, three separate matrices are utilized to represent the Red, Green, and Blue (RGB) channels, respectively:

$$I = [I_R \quad I_G \quad I_B] \tag{2}$$

$$I_k = \begin{bmatrix} p_{11}^{(k)} & p_{12}^{(k)} & \cdots & p_{1n}^{(k)} \\ p_{21}^{(k)} & p_{22}^{(k)} & \cdots & p_{2n}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1}^{(k)} & p_{m2}^{(k)} & \cdots & p_{mn}^{(k)} \end{bmatrix} \tag{3}$$

Each channel matrix I_k (where $k \in \{R, G, B\}$) encapsulates the intensity variations of the corresponding color component across the image. This matrix-based representation allows for efficient manipulation and processing of image data using linear algebraic techniques.

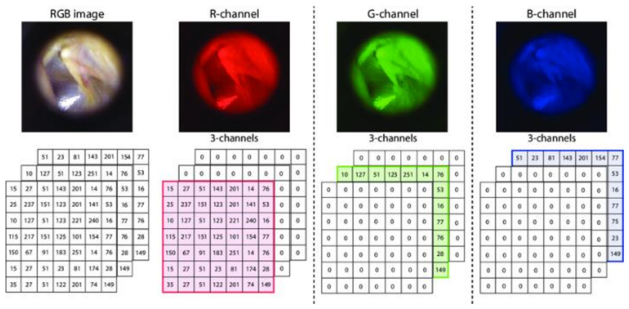


Figure 2.1 Matrix Representation of Digital Image https://www.researchgate.net/figure/Matrix-representation-of-a-digital-image-upper-row-from-left-to-right-image-in-the_fig2_359806413

C. Matrix Decomposition

A matrix decomposition is a way of reducing a complex matrix into its constituent parts which are in simpler forms [3]. Matrix decomposition refers to the factorization of a matrix into a product of matrices with specific properties, facilitating easier analysis and computation. Decomposition techniques are fundamental in simplifying complex matrix operations, solving linear systems, and performing dimensionality reduction. Common matrix decomposition methods include:

1. Eigen Decomposition—applicable to square matrices, eigen decomposition expresses a matrix as:

$$A = PDP^{-1} \tag{4}$$

where:
- P is a matrix, whose columns are the eigenvectors of A.
- D is a diagonal matrix containing the eigenvalues of A.
- P^{-1} is the inverse of matrix P.

Eigen decomposition is instrumental in understanding the intrinsic properties of linear transformations represented by matrices.

2. LU Decomposition—factors a matrix into a lower triangular matrix L and an upper triangular matrix U:

$$A = LU \tag{5}$$

This decomposition is widely used for solving linear systems and inverting matrices [3].

3. QR Decomposition—decomposes a matrix into an orthogonal matrix Q and an upper triangular matrix R:

$$A = QR \tag{6}$$

QR decomposition is essential in numerical methods, particularly in solving least squares problems [3].

4. Singular Value Decomposition (SVD) is a versatile decomposition applicable to any $m \times n$ matrix, which will be discussed in detail in the subsequent section.

Matrix decomposition techniques serve as foundational tools in various computational applications, enabling the transformation of complex matrix operations into more manageable forms.

D. Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a method for factorizing a matrix into three specific matrices: an orthogonal matrix, a diagonal matrix, and the transpose of another orthogonal matrix [4]. It is defined as follows, if A is an $m \times n$ matrix with rank k, then A can be decomposed into the following form:

$$A = U\Sigma V^T \tag{7}$$

$$U = [u_1 \quad u_2 \quad \cdots \quad u_k \quad u_{k+1} \quad \cdots \quad u_m] \tag{8}$$

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \sigma_k & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{9}$$

$$V^T = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_k^T \\ v_{k+1}^T \\ \vdots \\ v_n^T \end{bmatrix} \tag{10}$$

where U, Σ, and V have dimensions $m \times m$, $m \times n$, and $n \times n$, respectively, and the components satisfy the following properties [1]:
- $V = [v_1 \quad v_2 \quad \cdots \quad v_n]$ orthogonally diagonalizes $A^T A$.
- The nonzero diagonal entries of Σ are $\sigma_1 = \sqrt{\lambda_1}, \sigma_2 = \sqrt{\lambda_2}, \dots, \sigma_k = \sqrt{\lambda_k}$, where $\lambda_1, \lambda_2, \dots, \lambda_k$ are the nonzero eigenvalues of $A^T A$ corresponding to the column vectors of V.
- The column vectors of V are arranged such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0 \tag{11}$$

- The column vectors of U can be computed as

$$u_i = \frac{Av_i}{\| Av_i \|} = \frac{1}{\sigma_i} Av_i (i = 1,2, \dots, k) \qquad (12)$$

- The set $\{u_1, u_2, \dots, u_k\}$ forms an orthonormal basis for $\text{col}(A)$.
- The set $\{u_1, u_2, \dots, u_k, u_{k+1}, \dots, u_m\}$ extends the basis to form an orthonormal basis for R^m.

III. METHODOLOGY

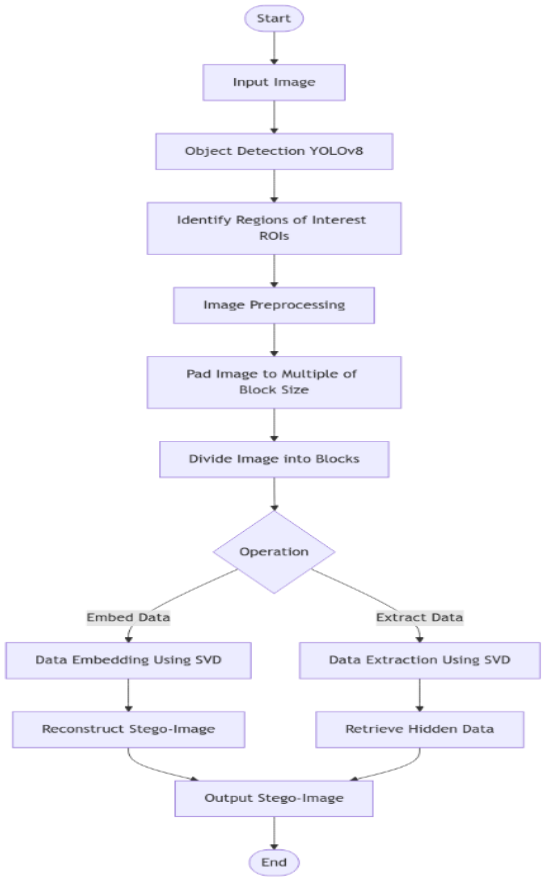


Figure 3.1 Flowchart of The Proposed Methodology Private Documentation

A. Object Detection for Region of Interest Identification

Object Detection is a pivotal component in identifying salient areas within an image that warrant focused processing. The utilization of the You Only Look Once version 8 (YOLOv8) algorithm facilitates real-time and accurate detection of objects, thereby delineating the ROIs for subsequent steganographic operations.

1. YOLOv8

YOLOv8 is a state-of-the-art object detection model characterized by its single-stage detection pipeline, which enables rapid and precise localization of objects within an image. Unlike traditional multi-stage detectors, YOLOv8 processes the entire image in a single forward pass, significantly reducing computational overhead while maintaining high detection accuracy.

The algorithm operates by partitioning the input image into a grid and predicting bounding boxes and class probabilities for each grid cell. These predictions are refined through successive convolutional layers, culminating in the identification of objects with their corresponding confidence scores and spatial coordinates.

2. Bounding Box Representation

A detected object is encapsulated by a bounding box, defined by the coordinates (x_1, y_1) and (x_2, y_2), representing the top-left and bottom-right corners of the rectangle surrounding the object, respectively. Mathematically, the bounding box can be expressed as:

$$\text{Bounding Box} = (x_1, y_1), (x_2, y_2) \qquad (13)$$

3. Region of Interest (ROI) Extraction

Upon detection, the bounding boxes corresponding to the specified object classes are extracted as ROIs. These ROIs serve as the focal points for data embedding, ensuring that hidden information is confined to areas of perceptual significance within the image. The precision of YOLOv8 in delineating ROIs enhances both the capacity and imperceptibility of the steganographic process.

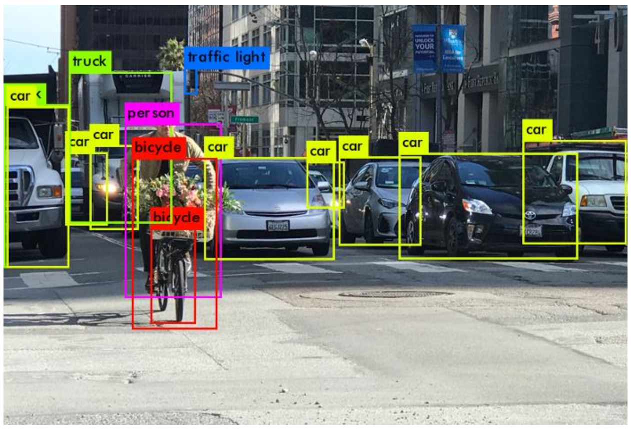


Figure 3.2 YOLO Image Detection Example
https://medium.com/analytics-vidhya/yolo-object-detection-made-easy-7b17cc3e782f

B. Image Preprocessing and Block Division

Effective steganography necessitates meticulous preprocessing of the host image to facilitate seamless data embedding. The preprocessing pipeline encompasses image padding and block division, ensuring compatibility with the SVD-based embedding mechanism.

1. Image Padding

To streamline the block division process, the host image is padded to dimensions that are multiples of the block size, typically 4×4 pixels. Padding mitigates boundary issues during block processing and maintains uniformity across all image segments.

Formally, given an image matrix I of dimensions $m \times n$, padding adjusts the dimensions to $m' \times n'$ such that:

$$m' = \left\lceil \frac{m}{4} \right\rceil \times 4, n' = \left\lceil \frac{n}{4} \right\rceil \times 4 \quad (14)$$

where $\lceil \cdot \rceil$ denotes the ceiling function.

2. Block Division

Post-padding, the image is partitioned into non-overlapping blocks of size 4×4 pixels. This segmentation facilitates localized processing, enabling discrete manipulation of image regions without inducing global distortions.

Mathematically, the image matrix I' of dimensions $m' \times n'$ is divided into $k = \frac{m'}{4} \times \frac{n'}{4}$ blocks, each represented as:

$$B_{\mathrm{ij}} = I'[4(i-1)+1:4i, 4(j-1)+1:4j] \quad (15)$$

$$\forall i \in \{1,2,\dots,\frac{m'}{4}\}, \forall j \in \{1,2,\dots,\frac{n'}{4}\}$$

C. Singular Value Decomposition for Data Embedding

Singular Value Decomposition (SVD) is leveraged as the core matrix decomposition technique for embedding hidden data within the image blocks. SVD decomposes each 4×4 image block into its constituent singular vectors and singular values, facilitating targeted modifications with minimal perceptual impact.

1. SVD Decomposition

For each image block B, SVD is performed as follows:

$$B = U\Sigma V^T \quad (16)$$

where:
- U is a 4×4 orthogonal matrix of left singular vectors.
- Σ is a 4×4 diagonal matrix containing singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \sigma_4 \geq 0$.
- V is a 4×4 orthogonal matrix of right singular vectors.

2. Data Embedding Procedure

The embedding process targets the least significant singular value σ_4 to encode binary data, capitalizing on its minimal influence on the visual integrity of the image block. The procedure entails the following steps:

- Binary Data Segmentation: The hidden message is converted into a binary sequence, with each bit corresponding to a modification in σ_4.
- Singular Value Perturbation:
  Bit '0' Embedding: If the bit to embed is '0', σ_4 remains unaltered.
  Bit '1' Embedding: If the bit is '1', a small perturbation Δ is added to σ_4:

$$\sigma_4' = \sigma_4 + \Delta \quad (17)$$

The perturbation Δ is carefully calibrated to ensure that the alteration is imperceptible while maintaining data integrity.

3. Block Reconstruction

The modified singular value matrix Σ' is utilized to reconstruct the stego-image block $B' = U\Sigma' V^T$

4. Block Replacement

The original image block B is replaced with the stego-image block B' in the overall image matrix.

D. Data Extraction Using Singular Value Decomposition

The extraction of hidden data necessitates the reversal of the embedding process, wherein the singular values of the stego-image blocks are analyzed to retrieve the embedded binary sequence.

1. SVD Decomposition for Extraction

For each stego-image block B', SVD is performed to obtain the $B' = U'\Sigma' V'^T$

2. Data Extraction Procedure

- Singular Value Analysis: The least significant singular value σ_4' is examined to determine the embedded bit.
- Bit Determination:
  Bit '0' Identification: If σ_4' remains unchanged or exhibits negligible perturbation, the corresponding bit is inferred as '0'.
  Bit '1' Identification: If a significant increase is detected in σ_4', the bit is inferred as '1'.
- Binary Sequence Reconstruction: The extracted bits are concatenated to reconstruct the original hidden message.

E. Image Assembly and Post-processing

Following the embedding or extraction of data within individual blocks, the image undergoes assembly and post-processing to restore its original dimensions and format.

1. Image Reconstruction

The modified blocks are systematically recombined to form complete stego-image matrix I'. Care is taken to ensure that the block alignment preserves the spatial coherence of the image.

2. De-padding

If padding was applied during preprocessing, it is subsequently removed to revert the image to its original dimensions. This ensures that the stego-image maintains consistency with the host image's spatial properties.

3. Format Conversion

The stego-image is saved in the desired format (currently only supports TIFF), with considerations for preserving image quality and embedded data integrity. Lossless format is a must to prevent data degradation.

F. Image Assembly and Post-processing

The methodology emphasizes balancing data embedding capacity with robustness against image processing operations. By confining data embedding to ROIs and targeting the least significant singular values, the framework ensures maximal data concealment with minimal perceptual distortion.

1. Capacity Analysis

The embedding capacity is determined by the number of available singular values within the identified ROIs. Specifically, each 4×4 block facilitates the embedding of one bit per color channel, yielding a cumulative capacity proportional to the number of blocks and color channels within the ROIs.

IV. Algorithm and Implementation

The system integrates object detection using YOLOv8 to identify Regions of Interest (ROIs) within an image, followed by image preprocessing, data embedding, and data extraction using Singular Value Decomposition (SVD). The implementation is executed in Python, leveraging essential libraries such as NumPy, OpenCV, Ultralytics YOLOv8, Pillow, and tifffile. Each component is comprehensively detailed through corresponding code snippets and thorough explanations to elucidate the operational workflow.

A. Object Detection for Region of Interest Identification

The initial phase of the implementation involves detecting objects within the input image to identify ROIs where hidden data will be embedded. YOLOv8, renowned for its precision and real-time performance, is employed to accurately delineate object boundaries, ensuring that data embedding is confined to relevant and visually significant regions.

```
from ultralytics import YOLO
import numpy as np

model = YOLO('yolov8n.pt')

def detect_objects(image_path, target_class):
 results = model(image_path)
 detections = []
 for result in results:
  for box in result.boxes:
   cls_id = int(box.cls[0])
   cls_name = model.names.get(cls_id, "Unknown")
   if cls_name.lower() == target_class.lower():
    x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
    detections.append({
     'class': cls_name,
     'x1': int(x1),
     'y1': int(y1),
     'x2': int(x2),
     'y2': int(y2)
    })
 return detections
```

The detect_objects function leverages the YOLOv8 model to perform object detection within the input image. Upon initializing the pre-trained YOLOv8 model (yolov8n.pt), the function processes the image specified by image_path. It iterates through the detection results, filtering objects that match the specified target_class. For each detected object, the function extracts the bounding box coordinates (x1, y1, x2, y2) and compiles them into a list of dictionaries. This list represents the ROIs where data embedding will subsequently occur. Utilizing YOLOv8's high accuracy in object detection ensures that data embedding is confined to relevant and visually significant regions, enhancing both the capacity and imperceptibility of the steganographic process.

B. Image Preprocessing

Post object detection, the identified ROIs undergo preprocessing to facilitate seamless data embedding. This involves padding the image to align with the block size requirements and dividing the image into non-overlapping blocks for localized processing.

1. Padding the Image

To ensure compatibility with block-wise processing, the image is padded such that both its height and width are multiples of the block size (e.g. 4×4 pixels). This uniformity prevents boundary discrepancies that could complicate data embedding and extraction.

```
def pad_to_multiple_of_block(image,
block_size=4):
height, width = image.shape[:2]
new_height = math.ceil(height / block_size) *
block_size
new_width = math.ceil(width / block_size) *
block_size

if len(image.shape) == 3:
 padded_image = np.zeros((new_height, new_width,
image.shape[2]), dtype=image.dtype)
else:
 padded_image = np.zeros((new_height,
new_width), dtype=image.dtype)

padded_image[:height, :width] = image
return padded_image
```

The pad_to_multiple_of_block function ensures that the input image's dimensions are divisible by the specified block_size. It calculates the necessary new dimensions by rounding up the original height and width to the nearest multiple of the block size using the ceiling function. Depending on whether the image is grayscale or color (determined by the number of dimensions), the function initializes a new array filled with zeros (representing black pixels) to accommodate the padded dimensions. The original image data is then copied into the top-left corner of this array, effectively adding the required padding without altering the original content. This preprocessing step is fundamental in maintaining the structural integrity of the image during the embedding process.

2. Dividing the Image into Blocks

Following padding, the image is segmented into non-overlapping blocks of size 4×4 pixels. This division facilitates localized data embedding and extraction, minimizing the risk of global distortions.

```
def divide_into_blocks(image, block_size=4):
 height, width = image.shape[:2]
 blocks = []
 for i in range(0, height, block_size):
  for j in range(0, width, block_size):
   block = image[i:i+block_size, j:j+block_size]
   blocks.append(((i, j), block))
 return blocks, height, width
```

The divide_into_blocks function systematically partitions the padded image into smaller, manageable blocks of 4 × 4 pixels. Iterating over the image dimensions in increments defined by the block_size, the function extracts each block and records its top-left coordinates. This organization is pivotal for mapping each block to its corresponding location within the image, thereby facilitating accurate data embedding and extraction. By focusing on these discrete blocks, the system can perform localized SVD operations, embedding data in a manner that minimizes disruption to the overall visual integrity of the image.

C. Data Embedding Using Singular Value Decomposition (SVD)

The embedding process utilizes SVD to decompose each 4 × 4 image block into singular vectors and singular values. Hidden data is then embedded by modifying the least significant singular values, ensuring minimal perceptual impact.

1. Embedding Data into ROIs

To embed the entire hidden message, the system iterates through each block within the identified ROIs, embedding corresponding bits sequentially.

```
def encode_message(image_path, message):
 I =
np.array(Image.open(image_path).convert('RGB'))
 I_padded = pad_to_multiple_of_block(I)
 I2 = I_padded.astype(np.float64) / 255.0

 M = message
 lm = len(M)
 MNum = [ord(char) for char in M]
 MNumFinal = [format(num, '08b') for num in
MNum]

 Emp = []
 for a in range(lm):
  for b in range(8):
   Emp.append(MNumFinal[a][b])
 Emp.append('2')  # End marker

 height, width = I2.shape[:2]
 encoded = np.copy(I2)

 isBreaking = False
 idx = 0

 for i in range(1, height // 4 + 1):
  for j in range(1, width // 4 + 1):
   for channel in range(3):
    block = I2[4*i-4:4*i, 4*j-4:4*j, channel]

    if block.shape[0] != 4 or block.shape[1] !=
4:
     continue

    U, s, VT = np.linalg.svd(block)
    S = np.zeros((4, 4))
    np.fill_diagonal(S, s)

    if idx < len(Emp):
     if Emp[idx] == '1':
      S[3, 3] = 0
     elif Emp[idx] == '0':
      if s[3] <= 1e-6:
       S[3, 3] = s[2] / 5
       if s[2] <= 1e-6:
        S[2, 2] = s[1] / 5
        S[3, 3] = S[2, 2] / 5
        if s[1] <= 1e-6:
         S[1, 1] = s[0] / 5
         S[2, 2] = S[1, 1] / 5
         S[3, 3] = S[2, 2] / 5
     elif Emp[idx] == '2':
      isBreaking = True
      break

     A = U @ S @ VT
     encoded[4*i-4:4*i, 4*j-4:4*j, channel] = A

    idx += 1
   if isBreaking:
    break
  if isBreaking:
   break

 encoded_cropped = encoded[:I.shape[0],
:I.shape[1]]
 encoded_cropped = np.clip(encoded_cropped, 0,
1.0)
 return encoded_cropped, I.shape[:2]
```

The encode_message function orchestrates the embedding of a hidden textual message within the host image using SVD-based steganography. Initially, the function loads the input image and normalizes its pixel values to the [0,1] range, ensuring numerical stability during SVD operations. The message is then converted into its binary representation, with each character translated into an 8-bit binary sequence. An end marker ('2') is appended to signify the termination of the embedded message during extraction.

The function proceeds to iterate over each 4 × 4 block and each colour channel (Red, Green, Blue) within the image. For each block, Singular Value Decomposition (SVD) is performed, decomposing the block into its singular vectors and singular values. The least significant singular value (σ_4) is then modified based on the current bit of the binary message:

- Bit '1': To embed a bit '1', the function sets the least singular value (S[3,3]) to zero. This modification encodes the bit within the block's singular value structure.
- Bit '0': Embedding a bit '0' involves a hierarchical adjustment. If the current least singular value (s[3]) is below a threshold (1×10^{-6}), it is modified by scaling down the preceding singular values. This process propagates upwards to maintain a balance between embedding capacity and perceptual quality. If the singular values are already low, further scaling ensures that the image's visual integrity remains intact.

If the end marker ('2') is encountered, the embedding process is terminated to prevent over-embedding. After modifying the singular values, the block is reconstructed using the altered singular value matrix and reassigned to its original position within the image array. This process continues sequentially for each bit in the binary message until the entire message, including the end marker, is embedded. Finally, the function crops the image back to its original dimensions, removing any padding added during preprocessing, and ensures that all pixel values remain within the valid [0,1] range. The result is an encoded image array with the hidden message seamlessly embedded within its singular values.

D. Data Extraction Using Singular Value Decomposition (SVD)

The extraction process reverses the embedding procedure, retrieving the hidden message by analyzing the least significant singular values of the stego-image blocks within the ROIs.

1. Extracting Data from ROIs

To retrieve the hidden message, the system iterates through each block within the ROIs, decoding corresponding bits sequentially.

```
def decode_message(encoded_image):
 height, width = encoded_image.shape[:2]
 NewM = []

 for i in range(1, height // 4 + 1):
  for j in range(1, width // 4 + 1):
   for channel in range(3):
    block = encoded_image[4*i-4:4*i, 4*j-4:4*j,
channel]
    if block.shape[0] != 4 or block.shape[1] !=
4:
     continue

    _, s, _ = np.linalg.svd(block)
    if np.abs(s[3]) <= 2e-7 - 5e-8:
     NewM.append('1')
    else:
     NewM.append('0')

 message = ""
 invalid_count = 0

 for k in range(0, len(NewM) - 7, 8):
  try:
   b = ''.join(NewM[k:k+8])
   a = int(b, 2)
   l = chr(a)

   if (not l.isprintable() or not (32 <= a <=
126)) and a != 10:
    invalid_count += 1
    if invalid_count >= 3:
     break
   else:
    invalid_count = 0
    message += l
  except:
   break

 return message
```

The decode_message function is designed to reverse the embedding process, extracting the hidden message from the stego-image by analyzing the least significant singular values of each 4×4 pixel block. The function begins by determining the dimensions of the encoded image and initializing an empty list (NewM) to accumulate the binary message.

Iterating over each block and each color channel, the function performs SVD on the block to obtain its singular values. The least significant singular value (σ_4) is then scrutinized to determine the embedded bit:

- Bit '1': If the absolute value of σ_4 is below a predefined threshold ($2 \times 10^{-7} - 5 \times 10^{-8}$), the function infers that a bit '1' was embedded.
- Bit '0': Otherwise, it deduces a bit '0'.

This binary sequence (NewM) accumulates progressively as the function traverses the entire image. Once the binary message is fully extracted, the function processes it in chunks of 8 bits to reconstruct individual characters. It converts each byte into its corresponding ASCII character and checks for the end marker ('00000010'), which signifies the termination of the embedded message. The function employs an invalid_count mechanism to detect consecutive invalid characters, ensuring that the extraction process halts appropriately to prevent the inclusion of corrupted or unintended data. The resultant message string is then returned, effectively recovering the original hidden information embedded within the stego-image.

E. Data Extraction Using Singular Value Decomposition (SVD)

Efficient image saving and loading are critical to preserving data integrity and ensuring seamless embedding and extraction processes. The implementation supports both TIFF format.

1. Saving the Image

The save_image function handles the export of processed images to disk in the desired format, ensuring that embedded data remains intact and that image quality is preserved.

```
from PIL import Image
import tifffile

def save_image(image, path, format="TIFF"):
 if format.upper() == "TIFF":
  try:
   with open('sRGB.icc', 'rb') as f:
    icc_profile = f.read()
  except FileNotFoundError:
   icc_profile = b''

  image_float32 = image.astype(np.float32)

  ICC_TAG_ID = 34675
  if icc_profile:
   icc_tag = (ICC_TAG_ID, 7, len(icc_profile),
icc_profile, True)
   extra_tags = [icc_tag]
  else:
   extra_tags = []

  with tifffile.TiffWriter(path, bigtiff=False)
as tiff_writer:
   tiff_writer.write(
    image_float32,
    photometric="rgb",
    extratags=extra_tags,
   )
```

The save_image function is responsible for persisting the processed image arrays to disk, based on user preference or application requirements. For TIFF format, the function attempts to embed the ICC profile (color profile) to preserve accurate color representation. If the ICC profile file (sRGB.icc) is not found, it proceeds without it, ensuring that the image saving process remains robust. The image array is cast to float32 to maintain high precision, which is particularly important for preserving subtle modifications in the singular values that encode the hidden message. The tifffile library's TiffWriter is employed to write the image data, specifying the photometric interpretation as RGB to correctly represent color information. This approach ensures that the stego-image retains its integrity and that the embedded data remains intact during storage, facilitating reliable extraction in subsequent operations.

2. Loading the Image

The load_image function facilitates the retrieval of images from disk, preparing them for embedding or extraction operations. This function supports both PNG and TIFF formats.

```
def load_image(path):
 if path.lower().endswith('.tiff') or
path.lower().endswith('.tif'):
  return tifffile.imread(path)
 else:
  image = Image.open(path).convert('RGB')
  return np.array(image).astype(np.float64) /
255.0
```

The load_image function is designed to import images from the filesystem, converting them into a standardized format suitable for further processing in the embedding or extraction pipelines. When dealing with TIFF files, the function leverages the tifffile.imread method to accurately read multi-dimensional data, which is essential for high-fidelity stego-images that may contain intricate embedded information. The pixel values are then cast to float64 and normalized to the [0,1] range, ensuring consistency with the preprocessing steps used during embedding.

For other image formats, such as PNG, the function utilizes PIL to open and convert the image to RGB format, facilitating uniform handling of color channels. The image data is subsequently converted into a NumPy array and normalized to the [0,1] range, aligning with the numerical requirements of the SVD-based embedding and extraction processes. This normalization is critical for maintaining the integrity of the embedded data and ensuring accurate reconstruction during extraction.

E. Reconstructing the Stego-Image and Retrieving Hidden Data

After embedding, the system reconstructs the stego-image by aggregating the modified blocks and ensuring the image retains its original dimensions and format. Conversely, during extraction, the system retrieves the hidden message by analyzing the reconstructed binary sequence.

1. Reconstructing and Saving the Stego-Image

The reconstruct_and_save_stego_image function finalizes the stego-image by cropping it to its original dimensions and saving it in the desired format.

```
def
reconstruct_and_save_stego_image(encoded_image,
original_dims, save_path='stego_image.png',
format='PNG'):
 encoded_cropped =
encoded_image[:original_dims[0],
:original_dims[1]]
 encoded_cropped = np.clip(encoded_cropped, 0,
1.0)

 stego_image_uint8 = (encoded_cropped *
255).astype(np.uint8)

 stego_image =
Image.fromarray(stego_image_uint8)
 stego_image.save(save_path, format=format)
```

The reconstruct_and_save_stego_image function finalizes the stego-image by cropping the encoded image array back to its original dimensions, removing any padding that was added during preprocessing. It ensures that all pixel values are clamped within the valid [0,1] range to maintain image integrity. The function then scales the normalized pixel values back to the [0,255] range and converts the data type to uint8, which is compatible with standard image formats. Utilizing the PIL library, the function creates an image object from the NumPy array and saves it in the specified format (e.g., PNG). This stego-image now contains the embedded hidden message within its specified ROIs, ready for storage or transmission.

2. Retrieving the Hidden Message

The retrieval process utilizes the previously defined extraction functions to decode the hidden message from the stego-image.

```
def retrieve_hidden_message(stego_image_path,
rois, block_size=4):
 stego_image = load_image(stego_image_path)
 extracted_message = decode_message(stego_image)
 return extracted_message
```

The retrieve_hidden_message function serves as the entry point for extracting the hidden message from a stego-image. It begins by loading the stego-image using the load_image function, which normalizes the image data and prepares it for analysis. The function then invokes the decode_message function, passing the loaded image and the list of ROIs to retrieve the embedded message. This modular approach ensures that each component of the extraction process operates cohesively, facilitating accurate and efficient retrieval of the concealed information. The final output is the reconstructed hidden message, effectively demonstrating the system's capability to securely embed and extract data within digital images.

V. Experiment Result

The experiments focus on visually demonstrating the system's capability to embed and accurately retrieve hidden messages within identified Regions of Interest (ROIs) while maintaining the visual integrity of the host images and quantitatively evaluate the embedding capacity.

A. Detection of Regions of Interest (ROIs)

The initial step involves detecting objects within the input images to identify ROIs suitable for data embedding. Utilizing YOLOv8, the system successfully identifies and outlines target objects, ensuring that data embedding is confined to relevant and visually significant areas.

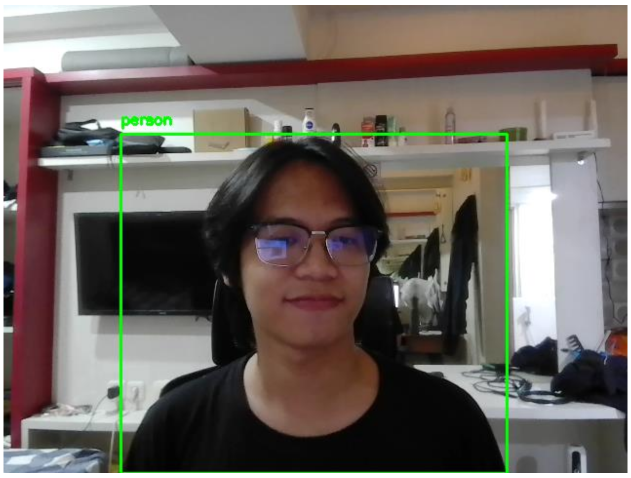


Figure 5.1 Detected ROI Image

B. Embedding Process: Before and After

The embedding process modifies the identified ROIs to conceal the hidden message. For this experiment, the hidden message is this:

test this is SECRET message for Mr. Rinaldi Munir and Mr. Rila Mandala
Sincere thanks are extended to the lecturers of ITB's Linear Algebra and Geometry IF2123 course, Mr. Rinaldi Munir and Mr. Rila Mandala
algeo2425{k3r3n_s3K4l1}

Below are visual comparisons of an image before and after the embedding process, demonstrating the system's ability to maintain the host image's visual quality.

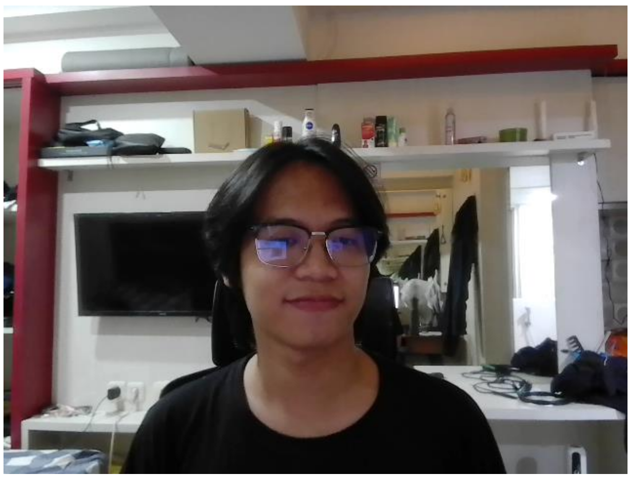


Figure 5.2 Original Image Before Embedding

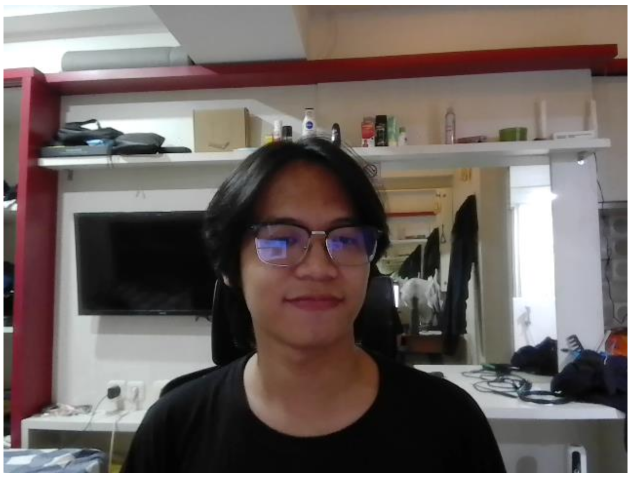


Figure 5.3 Image After Embedding Hidden Data

C. Visual Inspection

To assess the imperceptibility of the embedded data, zoomed-in sections of the ROIs are examined. These close-up views reveal minimal to no visible artifacts, confirming that the embedding process does not introduce noticeable defects from normal view.

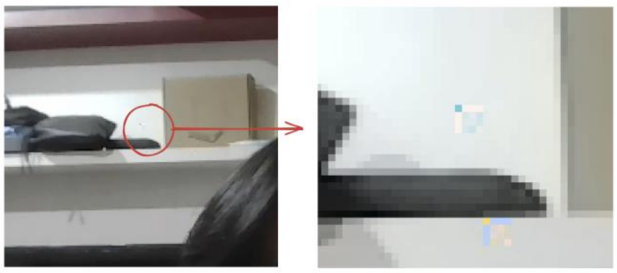
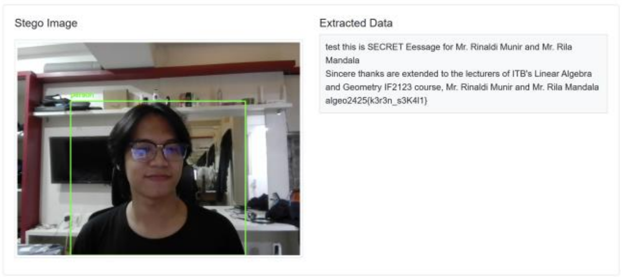


Figure 5.4 Zoomed-In View on Defected Area

A small defect can be seen as a weird 4×4 pixels spot. This defect is a result of modifying the block with a relatively high σ_4 value such that a modification can result in noticeable output.

D. Extraction and Verification of Hidden Data

This involves extracting the hidden message from the stego-image and verifying its accuracy against the original message. The extracted message is displayed on the screen.



test this is SECRET Eessage for Mr. Rinaldi Munir and Mr. Rila Mandala
Sincere thanks are extended to the lecturers of ITB's Linear Algebra and Geometry IF2123 course, Mr. Rinaldi Munir and Mr. Rila Mandala
algeo2425{k3r3n_s3K4l1}

Figure 5.5 Result of The Detected ROI and Extracted Data

Extracted Message:

test this is SECRET Eessage for Mr. Rinaldi Munir and Mr. Rila Mandala
Sincere thanks are extended to the lecturers of ITB's

```
Linear Algebra and Geometry IF2123 course, Mr.
Rinaldi Munir and Mr. Rila Mandala
algeo2425{k3r3n_s3K4l1}
```

The extracted message matches the original exactly, confirming the effectiveness of the embedding and extraction processes.

E. Embedding Capacity

The table below summarizes the image resolutions, corresponding ROI sizes, and the embedding capacities achieved during the experiments. It demonstrates that higher-resolution images with larger ROIs allow for embedding a greater number of characters, enhancing the system's capacity to conceal more information. Conversely, smaller ROIs in lower-resolution images result in reduced embedding capacity, highlighting the system's adaptability based on image and ROI dimensions.

TABLE 1
Summary of Image Resolutions, ROI Sizes, and Embedding Capacities.

Image Resolution (Pixels)	**ROI Size (Pixels)**	**Embedding Capacity (Characters)**
480 x 640	408 x 336	3,213
480 x 640	264 x 228	1,410
833 x 658	96 x 192	432
1,200 x 1,200	684 x 1,044	16,736

VI. CONCLUSION

This study successfully developed an image steganography system that integrates Singular Value Decomposition (SVD) with YOLOv8 for identifying Regions of Interest (ROIs) within images. By leveraging YOLOv8's precise object detection capabilities, the system effectively targets specific areas for data embedding, ensuring that hidden messages are concealed without compromising the visual quality of the host images.

The experimental results demonstrate the system's ability to embed and accurately retrieve hidden messages with minimal perceptual impact. Visual inspections confirmed that the stego-images maintained high fidelity, with no noticeable artifacts, while the extraction process reliably recovered the embedded information. This highlights the effectiveness of combining advanced object detection with matrix decomposition techniques for secure and imperceptible data concealment.

While the system shows strong performance in controlled environments, future work could explore enhancing embedding capacity and robustness against more sophisticated image manipulations. Additionally, integrating encryption methods could further secure the concealed data, providing an extra layer of protection. Overall, the integration of YOLOv8 and SVD presents a promising approach to advancing the field of image steganography.

VII. APPENDIX

The source code for the implementation discussed in this paper is available at the following GitHub repository: https://github.com/l0stplains/Steganography-ROI-SVD

VIII. ACKNOWLEDGMENT

The author wishes to express heartfelt gratitude to God for His guidance and support throughout the completion of this project. Sincere thanks are extended to the lecturers of ITB's Linear Algebra and Geometry IF2123 course, Mr. Rinaldi Munir and Mr. Rila Mandala, for their invaluable knowledge and mentorship. Additionally, the author is deeply thankful to family and friends for their unwavering encouragement and support during this endeavor and chaotic end of semester. Their continuous support served as a foundation for overcoming challenges and achieving the objectives of this study.

REFERENCES

[1] H. Anton and A. Kaul, *Elementary Linear Algebra*, 12th ed, Hoboken, NJ: Wiley, 2019.

[2] O. Ogunleye and O. Iyiola, *Application of Matrix Theory to Image Processing*. ResearchGate, Dec. 2021. [Online]. Available: https://www.researchgate.net/publication/379666441_Application_of_Matrix_Theory_to_Image_Processing. Accessed: Dec. 28, 2024.

[3] J. Lu, *Matrix Decomposition and Applications*. arXiv, Jan. 2022. [Online]. Available: https://www.researchgate.net/publication/357553092_Matrix_Decomposition_and_Applications. Accessed: Dec. 28, 2024.

[4] R. Munir, *Singular Value Decomposition (SVD) (Bagian 1)*, Lecture Notes for IF2123 Algebra Linier dan Geometri, Program Studi Teknik Informatika, STEI-ITB, 2023–2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf. Accessed: Dec. 28, 2024.

DECLARATION OF ORIGINALITY

I hereby declare that the paper I have written is entirely my own work, and not a reproduction, adaptation, or translation of another individual's work. Furthermore, I declare that this paper is free from any form of plagiarism..

Bandung, 2 January 2025

[illegible]

Refki Alfarizi
13523002