Image Encryption and Decryption Using Chaotic Sequences and VGG16 Features

May 19, 2025

1 Aim

The primary objective of this project is to develop and evaluate a novel image encryption and decryption system that leverages chaotic sequences derived from a three-dimensional hyperchaotic map and feature extraction using the VGG16 convolutional neural network. The system aims to provide a secure method for protecting image data by scrambling pixel positions using chaotic permutations, ensuring that the encrypted image is visually indistinguishable from noise. Additionally, the system incorporates VGG16 feature extraction to explore potential enhancements in the encryption process, although the features are not directly utilized in the current implementation. The decryption process aims to perfectly reconstruct the original image using the inverse permutation derived from the same chaotic sequence, demonstrating the reversibility and reliability of the proposed method.

The specific goals include:

- Implementing a chaotic sequence generator based on a three-dimensional hyperchaotic map with predefined parameters.
- Developing an encryption algorithm that shuffles image pixels according to the chaotic sequence.
- Creating a decryption algorithm that reverses the permutation to recover the original image.
- Integrating VGG16 for feature extraction to lay the groundwork for future enhancements.
- Evaluating the system's performance in terms of encryption quality, decryption accuracy, and computational efficiency.

This approach combines the unpredictability of chaotic systems with the robustness of deep learning feature extraction, aiming to create a secure and reversible image encryption scheme suitable for applications requiring data confidentiality.

2 Methodology

The methodology involves a multi-step process that integrates image processing, chaotic sequence generation, and deep learning feature extraction. The system is implemented in Python using libraries such as NumPy, PIL (Python Imaging Library), and TensorFlow/Keras for VGG16 integration. The following subsections detail the key components of the methodology.

2.1 Chaotic Sequence Generation

A three-dimensional hyperchaotic map is employed to generate pseudo-random sequences for pixel shuffling. The map is defined by the following iterative equations:

$$x_{n+1} = (x_n + a_1 x_n + a_2 y_n + a_3 y_n^2) \mod 1, \tag{1}$$

$$y_{n+1} = (y_n + b_1 - b_2 z_n) \mod 1, \tag{2}$$

$$z_{n+1} = (z_n + c x_n) \mod 1, \tag{3}$$

where the parameters are set as $a_1 = 0.05$, $a_2 = 0.25$, $a_3 = 0.12$, $b_1 = 4$, $b_2 = 1.2$, and $c = 2.15$. The initial conditions are $x_0 = y_0 = z_0 = 0.1$. These parameters ensure the map exhibits chaotic behavior, producing sequences that are highly sensitive to initial conditions and suitable for cryptographic applications.

The function `generate_chaotic_sequence(length)` iterates the map to produce sequences X, Y, and Z, each of length equal to the total number of pixels in the image. The sequence X is sorted to generate permutation indices, which are used to shuffle the image pixels.

2.2 Image Preprocessing

The input image is loaded using PIL and converted to RGB format to ensure consistency. The image is represented as a three-dimensional NumPy array with dimensions (height, width, channels). For encryption, the array is flattened into a two-dimensional array of shape (height $\times$ width, channels) to facilitate pixel shuffling. Additionally, the image is resized to 224×224 pixels for VGG16 feature extraction, as required by the model's input layer.

2.3 VGG16 Feature Extraction

The VGG16 model, pre-trained on the ImageNet dataset, is used to extract high-level features from the resized image. The model is configured to output features from the `block5_pool` layer, resulting in a flattened feature vector. The image is preprocessed using the `preprocess_input` function to normalize pixel values according to VGG16 requirements. While the extracted features are not currently used in the encryption process, they are computed to explore potential future applications, such as feature-based key generation or image authentication.

2.4 Encryption Process

The encryption algorithm shuffles the image pixels based on the chaotic sequence:

1. Compute the chaotic sequence X for a length equal to the total number of pixels (height $\times$ width).
2. Sort X to obtain permutation indices (`perm_indices`).
3. Apply the permutation to the flattened image array, rearranging the pixels.
4. Store the shuffled pixels and image dimensions globally for decryption.
5. Reshape the shuffled pixels back to the original image dimensions and save the encrypted image as a JPEG or PNG file.

This process ensures that the encrypted image appears as random noise, obscuring the original content.

2.5 Decryption Process

The decryption algorithm reverses the permutation to recover the original image:

1. Regenerate the chaotic sequence X using the same parameters and initial conditions.
2. Compute the permutation indices and their inverse (`reverse_perm_indices`).
3. Apply the inverse permutation to the stored shuffled pixels.
4. Reshape the resulting pixels to the original image dimensions and save the decrypted image.

The use of identical chaotic parameters ensures perfect reconstruction, as the permutation is deterministic and reversible.

2.6 Implementation Environment

The system is implemented in a Google Colab environment, leveraging its support for file uploads and downloads. The following libraries are used:

- `numpy` for numerical computations and array manipulation.
- `PIL` for image loading and saving.
- `tensorflow.keras` for VGG16 model integration.
- `google.colab.files` for handling file uploads and downloads.

The encrypted and decrypted images are saved to the `/content` directory and downloaded for evaluation.

3 Results

The proposed image encryption and decryption system was successfully implemented and tested. The following subsections summarize the key results, including encryption quality, decryption accuracy, and computational considerations.

3.1 Encryption Quality

The encryption process effectively transforms the input image into a visually unrecognizable form. The chaotic permutation scatters the pixels randomly, resulting in an encrypted image that resembles noise. Visual inspection confirms that no discernible patterns or structures from the original image remain in the encrypted output. This indicates a high level of security, as the encrypted image provides no immediate information about the original content.

3.2 Decryption Accuracy

The decryption process successfully reconstructs the original image with perfect accuracy. By applying the inverse permutation derived from the same chaotic sequence, the system restores the pixel positions exactly as they were in the original image. Pixel-wise comparison between the original and decrypted images shows zero differences, confirming that the decryption is lossless. This reversibility is a critical feature for practical applications, ensuring that no data is lost during the encryption-decryption cycle.

3.3 VGG16 Feature Extraction

The VGG16 model successfully extracts features from the resized input image, producing a flattened feature vector. For a 224×224 RGB image, the output from the `block5_pool`

layer is a tensor of shape $(7, 7, 512)$, which is flattened to a vector of length 25,088. While these features are not currently used in the encryption or decryption process, their successful extraction demonstrates the system's compatibility with deep learning frameworks. Future work could explore incorporating these features into the chaotic map parameters or using them for additional security measures, such as image authentication.

3.4 Computational Performance

The system performs efficiently within the Google Colab environment. The chaotic sequence generation is computationally lightweight, as it involves simple iterative calculations. The VGG16 feature extraction is the most computationally intensive step, requiring approximately 12 seconds on a standard Colab GPU runtime. The encryption and decryption processes are fast, with pixel shuffling taking less than a second for typical image sizes (e.g., 512×512 pixels). The use of NumPy for array operations ensures efficient handling of large image data.

3.5 Limitations and Observations

One limitation is the reliance on global variables to store the shuffled pixels and image dimensions for decryption. This approach assumes that encryption and decryption occur within the same session, which may not be practical for real-world applications. Additionally, the VGG16 features are currently unused in the encryption process, limiting their immediate impact. The system also assumes that the chaotic map parameters and initial conditions are securely shared between the encryption and decryption parties, which is a critical consideration for practical deployment.

3.6 Future Enhancements

Future work could address the following:

- Incorporate VGG16 features into the chaotic map parameters to create a content-dependent encryption key.
- Implement a secure key exchange mechanism to share chaotic map parameters.
- Optimize the system for larger images and real-time applications.
- Explore alternative chaotic maps or deep learning models for enhanced security.

4 Conclusion

The developed image encryption and decryption system successfully achieves its objectives of securing image data using chaotic sequences and integrating VGG16 feature extraction. The chaotic map provides a robust mechanism for pixel shuffling, resulting in a visually secure encrypted image. The decryption process is lossless, ensuring perfect reconstruction of the original image. While VGG16 features are not yet utilized in the encryption process, their successful extraction opens avenues for future enhancements. The system demonstrates a promising combination of chaotic systems and deep learning, with potential applications in secure image communication and storage.